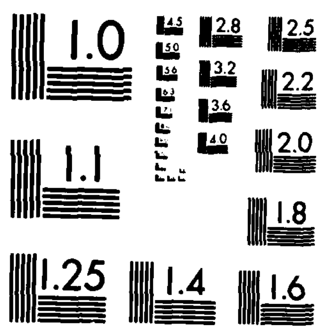
AD-A153 609 EVALUATION AND VALIDATION (E&V) TEAM PUBLIC REPORT VOLUME 1(U) AIR FORCE WRIGHT AERONAUTICAL LABS WRIGHT-PATTERSON AFB OH V L CASTOR 30 NOV 84 1/6

UNCLASSIFIED AFWAL-TR-85-1016-VOL-1 F/G 14/2 NL


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0

(2)

AFWAL TR 85-1016

AD-A153 609

EVALUATION AND VALIDATION (E&V)

TEAM PUBLIC REPORT

Volume I

VIRGINIA L. CASTOR
E&V TEAM CHAIRPERSON
AVIONICS LABORATORY
AIR FORCE WRIGHT AERONAUTICAL LABORATORIES
WRIGHT-PATTERSON AIR FORCE BASE, OHIO 45433-6543

30 NOVEMBER 1984

Interim Technical Report for Period
1 October 1983-30 September 1984

PREPARED FOR:

ADA JOINT PROGRAM OFFICE
3D139 (FERN ST/C107) PENTAGON
WASHINGTON, D.C. 20301

NOTICE

When Government drawings, specifications, or other data are used for any purpose other than in connection with a definitely related Government procurement operation, the United States Government thereby incurs no responsibility nor any obligation whatsoever; and the fact that the government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data, is not to be regarded by implication or otherwise as in any manner licensing the holder or any other person or corporation, or conveying any rights or permission to manufacture use, or sell any patented invention that may in any way be related thereto.

This report has been reviewed by the Office of Public Affairs (ASD/PA) and is releasable to the National Technical Information Service (NTIS). At NTIS, it will be available to the general public, including foreign nations.

This technical report has been reviewed and is approved for publication.

Virginia L. Castor

Virginia L. Castor
Project Engineer

Date 22 Jan 85

FOR THE COMMANDER

Raymond D. Bellem

RAYMOND D. BELLEM, COL, USAF
Deputy Chief
System Avionics Division
Avionics Laboratory

22 Jan 85
Date

"If your address has changed, if you wish to be removed from our mailing list, or if the addressee is no longer employed by your organization please notify AFWAL/AAAF-2, W-PAFB, OH 45433 to help us maintain a current mailing list."

Copies of this report should not be returned unless return is required by security considerations, contractual obligations, or notice on specific document.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) AFWAL-TR-85-1016	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION AIR FORCE WRIGHT AERONAUTICAL LABORATORIES 6b. OFFICE SYMBOL (If applicable) AAAF	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State and ZIP Code) WRIGHT-PATTERSON AFB OHIO 45433-6543	7b. ADDRESS (City, State and ZIP Code)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Ada JOINT PROGRAM OFFICE 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State and ZIP Code) 3D139 (Fern St/C107) Pentagon Washington, DC 20301	10. SOURCE OF FUNDING NOS.

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT NO.
63226	AJPO	28	53

11. TITLE (Include Security Classification) Evaluation and Validation (E&V) Team Public Report, Volume I

12. PERSONAL AUTHOR(S)
Virginia L. Castor, E&V Team Chairperson

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Yr., Mo., Day)	15. PAGE COUNT
Interim	FROM 1 Oct 83 TO 30Sep84	1984 November 30	494

16. SUPPLEMENTARY NOTATION
*Ada is a Registered Trademark of the U.S. Government (Ada Joint Program Office)

17. COSATI CODES

FIELD	GROUP	SUB. GR.

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
Evaluation
Validation
Ada*
Programming Support Environments

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

The initial activities and accomplishments of the Evaluation and Validation (E&V) Team are reported. The purpose of the E&V Task, which is sponsored by the Ada Joint Program Office (AJPO), is to develop the techniques and tools which will provide a capability to perform assessment of Ada Programming Support Environments (APSEs) and to determine conformance of APSEs to the Common APSE Interface Set (CAIS). As this technology is developed, it is being made available to DoD components, industry and academia. As with all Ada-related activities, the widest possible participation in the E&V Task is encouraged.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS ☐	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED

22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE NUMBER (Include Area Code)	22c. OFFICE SYMBOL
Virginia L. Castor	(513) 255-2446	AFWAL/AAAF

DD FORM 1473, 83 APR EDITION OF 1 JAN 73 IS OBSOLETE.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

SECTION I

PROJECT TECHNICAL SUMMARY

1.1 Introduction

This report is the first in a series of annual technical reports to be published by the Evaluation and Validation (E&V) Team. The purpose of the E&V Public Report is to provide an overview of the many technical accomplishments of the E&V Team during the preceding fiscal year. This first report contains information resulting from E&V activities during fiscal year 1984 (October 1983 - September 1984) which is being made available for public review and comment. Contents of this report r flect an observation of the E&V Team progress during the fiscal year and should not be viewed as final representations of the technology being developed.

1.2 Background

In June 1983 the Ada Joint Program Office (AJPO) proposed the formation of the E&V Task and a tri-service E&V Team, with the Air Force designated as lead service. The purpose of the E&V Task is to develop the techniques and tools which will provide a capability to perform assessment of Ada Programming Support Environments (APSEs) and to determine conformance of APSEs to the Common APSE Interface Set (CAIS). As the E&V technology is developed, it will be made available to the community for use by DoD components, industry, and academia as deemed appropriate by the respective organizations. In October 1983 the Air Force officially accepted responsibility as lead service and designated the Air Force Wright Aeronautical Laboratories (AFWAL) at Wright-Patterson Air Force Base as lead organization. By November 1983, a comprehensive E&V Plan was developed, and by December 1983 an E&V Team had been established, with the first meeting held at Wright-Patterson Air Force Base. In April 1984, an E&V Workshop was held at Airlie, Virginia. The purpose of the Workshop was to solicit the participation of industry representatives in the E&V Task. Many of the participants in the E&V Workshop have chosen to remain involved in the E&V Task as Distinguished Reviewers, and have contributed significantly to the accomplishments of the E&V Team.

1.3 E&V Meetings

E&V Team meetings are held quarterly, and all E&V Team meetings during the last fiscal year were held at Wright-Patterson Air Force Base (7-8 December 1983, 7-8 March 1984, 6-8 June 1984, and 5-7 September 1984). The E&V Workshop, which is scheduled on an annual basis, was held in Airlie, Virginia 2-6 April 1984. Communication among E&V Team members throughout the year is accomplished primarily via the MILNET.

1.4 E&V Team Organization

The E&V Team is organized into the following five working groups:

a. Requirements Working Group (REQWG)

The REQWG is responsible for reviewing life-cycle methodology materials to determine life-cycle issues which should be addressed by the E&V Team; developing an E&V Requirements Document; refining E&V requirements outputs from the E&V Workshop; providing analysis of E&V requirements to determine completeness, traceability, testability, consistency, and feasibility; identifying issues which may impact the development of E&V technology but which do not necessarily correlate to APSE components; and providing recommendations for development/acquisition of E&V tools/aids.

b. Technical Coordination Working Group (TECWG)

The TECWG is responsible for performing a literature search for efforts relevant to the E&V Task; developing a Technical Coordination Strategy Document which documents the relationship of these efforts to the E&V Task; and providing technical presentations to the E&V Team on these related efforts.

c. APSE Analysts Working Group (APSEWG)

The APSEWG is responsible for providing expertise on DoD and commercial APSEs available within the DoD; providing presentations to the E&V Team on these APSEs; identifying existing capabilities/tests/tools associated with each APSE; developing a DoD APSE Analysis Document; and monitoring DoD APSE Formal Qualification Testing.

d. Common APSE Interface Set Working Group (CAISWG)

The CAISWG is responsible for providing expertise on the CAIS; providing presentations to the E&V Team on the CAIS; providing liaison activities with the KIT/KITIA; recommending specific areas of consideration for the CAIS Validation Capability contractual effort; and developing an APSE Validation Procedures Document.

e. Public Coordination Working Group (PUBWG)

The PUBWG is responsible for identifying professional organizations which are technically related to the E&V Task; developing a Public Coordination Strategy Document; recording minutes of all E&V Team meetings; preparing E&V Status Reports; and developing and maintaining an E&V project reference list.

1.5 Document Organization

This document is organized as follows:

a. Appendix A - Evaluation and Validation (E&V) Plan

The purpose of the E&V Plan is to provide a detailed and organized approach to the development of technology which will be used as a basis for the E&V of APSEs. The E&V Plan which is included in this document was developed in November 1983 and was used as a basis for technical guidance to the E&V Team throughout fiscal year 1984. The E&V Plan will be updated annually.

b. Appendix B - Requirements for Evaluation and Validation of Ada Programming Support Environments

The purpose of the E&V Requirements Document is to set forth requirements on the E&V effort; i.e., requirements against which the organization and activities of the E&V Team can be mapped, requirements on the E&V methods and procedures, and requirements on what is to be evaluated within an APSE. This document was developed primarily by the REQWG and it will be updated at intervals specified within the E&V Plan.

c. Appendix C - DoD APSE Analysis Document

The purpose of the DoD APSE Analysis Document is to identify existing DoD APSEs and to provide a taxonomy of the capabilities of each of these environments. The environments identified in this document include the Air Force's Ada Integrated Environment (AIE), the Army's Ada Language System (ALS), and the Navy's Ada Language System/Navy (ALS/N). This document was developed primarily by the APSEWG and it will be updated at intervals specified within the E&V Plan. Future versions will include a comparison of the functional capabilities of the identified environments, the evaluation criteria, and an analysis of the application of the criteria to the existing DoD APSEs.

d. Appendix D - Evaluation and Validation Technical Coordination Strategy Document

The purpose of the E&V Technical Coordination Strategy Document is to identify existing efforts/organizations which are technically related to the E&V Task, to identify the relationships, to identify areas of mutual benefit, to identify impact of schedules, to identify the level of coordination which is required, and to identify issues which require resolution among tasks. This document was developed primarily by the TECWG and it will be updated at intervals specified within the E&V Plan.

e. Appendix E - Evaluation and Validation Public Coordination Strategy Document

The purpose of the E&V Public Coordination Strategy Document is to facilitate the transition of E&V technology to the public by identifying appropriate professional organizations and points of contact to be notified, as appropriate, of progress within the E&V Task. This document was developed primarily by the PUBWG and it will be updated at intervals specified within the E&V Plan.

f. Appendix F - Minutes of the Evaluation and Validation (E&V) Meeting 7-8 December 1983

The detailed minutes of the first E&V Team meeting include an overview of presentations by guest speakers Major Israel Caro (AFWAL/AAAF), who discussed the Air Force perspective of E&V; LCDR Brian Schaar (AJPO), who discussed the AJPO's tasking philosophy; John Kramer (Institute for Defense Analyses), who provided a presentation on the philosophy of environments and who also discussed the status of the Kernal APSE (KAPSE) Interface Team; and finally, Robert Knapper (Institute for Defense Analyses), who discussed Ada compiler validation procedures and lessons learned.

g. Appendix G - Minutes of the Evaluation and Validation (E&V) Meeting 7-8 March 1984

The detailed minutes of the second E&V Team meeting include an overview of presentations by guest speakers Patricia Oberndorf (NOSC), who provided background information and the current status of the KIT/KITIA; Timothy Lindquist (Virginia Polytechnic Institute and State University), who presented an overview of an AJPO-sponsored effort for developing a specification technique for CAIS; and Ronnie Martin (Georgia Institute of Technology), who described the Software Test and Evaluation Project (STEP) which was initiated by the Director Defense Test and Evaluation.

h. Appendix H - Minutes of the Evaluation and Validation (E&V) Meeting 6-8 June 1984

The detailed minutes of the third E&V Team meeting include an overview of presentations by guest speakers Elizabeth Bailey (Institute for Defense Analyses), who provided information on evaluating APSE usability; Charles McKay (University of Houston), who discussed the Johnson Space Center APSE Project; and Raymond Szymanski (AFWAL/AAAF), who described the Integrated Support Software System environment under development at AFWAL.

i. Appendix I - Minutes of the Evaluation and Validation (E&V) Meeting 5-7 September 1984

The detailed minutes of the fourth E&V Team meeting include an overview of presentations by guest speakers Captain Ricardo Contreras (HQ AFOTEC), who discussed the Air Force Operational Test and Evaluation Center activities; and Richard Drake (IBM), who discussed a project focused on independent testing of an Ada compiler in support of the Submarine Advanced Combat System (SUBACS).

j. Appendix J - Evaluation Criteria for Ada Compilers

This document was developed by Elizabeth Kean, APSEWG Chairperson, in support of the E&V Task. It contains a list of evaluation criteria for Ada compilers. The criteria and associated identified Ada tests are designed to complement the Ada Compiler Validation Capability (ACVC).

k. Appendix K - E&V Workshop Position Papers

This appendix contains position papers which were submitted in response to solicitation to industry for participation in the April 1984 E&V Workshop in Airlie, Virginia. The authors whose papers are included in this document include: Bard Crawford (The Analytic Sciences Corporation), Paul Dobbs (General Dynamics), Robert Fritz (Computer Sciences Corporation), Kathleen Gilroy (Harris Corporation), Kathleen Gracy (SofTech, Inc.), Charles Hammons (Texas Instruments, Inc.), Asha Kant (Litton Applied Technology), Robert Kirkpatrick (Data General), Susan Mickel (General Electric Company), James Parlier (General Dynamics), John Reddan (Syscon Corporation), Amos Rohrer (EG&G, WASCI), Helen Romanowsky (Rockwell International), Andres Rudmik (GTE Communication Systems), Raymond Sandborgh and Michael Meirink (Sperry Corporation), Paul Scheffer (Martin Marietta Denver Aerospace), and James Winchester (Hughes Aircraft Company).

1.6 Conclusion

This E&V Public Report is being made available by the E&V Team in order to solicit comments from those individuals who are not actively involved in the E&V Task. All comments should be addressed to:

Virginia L. Castor
AFWAL/AAAF
Wright-Patterson Air Force Base
Ohio 45433-6543.

Arpanet Address: CASTOR@USC-ECLB

Appendix A

Evaluation and Validation
(E&V)

Plan

Version 1.0

30 November 1983

Table of Contents

Table of Contents (Continued)

1. INTRODUCTION

1.1 Objective of the E&V Plan

The purpose of the E&V Plan is to provide a detailed and organized approach to the development of technology which will be used as a basis for the Evaluation and Validation (E&V) of Ada Programming Support Environments (APSEs). The E&V Plan will be updated on an annual basis throughout the duration of the E&V Task.

This document is organized as follows:

- Section 1: INTRODUCTION

    * Section 1 presents: (1) the objective of the E&V Plan; and (2) historical background information which led to the establishment of the E&V Team.

- Section 2: SCOPE

    * Section 2 presents the scope of the E&V Task through delineation of the E&V Task objectives.

- Section 3: E&V TECHNICAL APPROACH

    * Section 3 provides an overview of the technical approach to the development of E&V technology by defining an initial E&V Classification Schema.

- Section 4: E&V MANAGEMENT APPROACH

    * Section 4 provides the management structure for the E&V Task and identifies specific tasks for Working Groups within the E&V Team.

- Section 5: E&V RELATIONSHIP TO OTHER ORGANIZATIONS

    * Section 5 describes the relationship of the E&V Task to other DoD and technical organizations.

- Section 6: E&V DELIVERABLES

    * Section 6 presents a description of all of the deliverables expected from the E&V Task.

- Section 7: E&V WORK BREAKDOWN STRUCTURE

    * Section 3 presents a Work Breakdown Structure which delineates all of the activities to be accomplished in the E&V Task.

- Section 8: E&V SCHEDULES/MILESTONES

    * Section 8 presents schedules and milestones associated with the E&V Task.

- Section 9: E&V REFERENCES

    * Section 9 provides a list of references which are used within this document.

1.2 Background

In 1975 the High Order Language Working Group (HOLWG) was formed under the auspices of the U.S. DoD. It consisted of representatives from the Army, Air Force, Navy, Marines and other defense agencies, with the goal of establishing a single high order language for new DoD Embedded Computer Systems (ECS). The technical requirements for the common language were finalized in the STEELMAN [1] report of June 1978. International competition was used to select the new common language design. In 1979, after review by approximately eighty teams (representing DoD organizations, industry, academia and NATO countries), and after intensive analysis by the three Services and other defense agencies, the DoD selected the design developed by Jean Ichbiah and his colleagues at CII-Honeywell Bull. The language was named Ada in honor of Agusta Ada Byron (1815-1851), the daughter of Lord Byron and the first computer programmer.

Early in the development process it was realized that the acceptance and the benefits derived from a common language could be increased substantially by the development of an integrated system of software development and maintenance tools. The requirements for such an Ada programming environment were stated in the STONEMAN [2] document. STONEMAN identifies the APSE as support for "the development and maintenance of Ada application software throughout its life cycle."

The Army and Air Force have begun separate developments of APSEs; the Navy is in the process of procuring an APSE development, which will be based upon the Army's APSE. The Army APSE has been designated the Ada Language System (ALS); the Air Force APSE has been designated the Ada Integrated Environment (AIE); and the Navy APSE has been designated the Ada Language System/Navy (ALS/N).

The Ada Joint Program Office (AJPO) was formed in December 1980. It is the principal DoD agent for development, support and distribution of tools, common libraries, and coordination of Ada. The AJPO will coordinate all Ada efforts within DoD to ensure their compatibility with the requirements of other Services and DoD agencies, to avoid duplicative efforts and to maximize sharing of resources.

The KAPSE Interface Team (KIT), a tri-serivce organization which is chaired by the Navy under the guidance of the AJPO, was established in late 1981 as the result of a Memorandum of

Agreement (MOA) signed by the Deputy Under Secretary of Defense and the Assistant Secretaries of the three services. The objective of the KIT is to define a standard set of Kernel Ada Programming Support Environment (KAPSE) interfaces to ensure the interoperability of data and the transportability of tools between conforming APSEs. The Common APSE Interface Set (CAIS) developed by the KIT provides the virtual operating system on which tools run, as well as the minimum set of command, edit and similar functions required to transport tools from one CAIS to another. The KAPSE Interface Team from Industry and Academia (KITIA) was established in early 1982. The KITIA consists of volunteer representatives from industry and universities who provide expertise relevant to the definition of the CAIS.

In addition to the KIT/KITIA development of the CAIS, which will require the development of a validation capability to determine conformance, other efforts have contributed to the foundation of the E&V Task. One such effort was the formation of the Ada Validation Organization (AVO), under the direction of the AJPO. The AVO is responsible for the development of an Ada Compiler Validation Capability (ACVC) which is currently used to ensure that Ada compiler developers have correctly implemented the standard Ada language (ANSI/MIL-STD-1815A-1983). A second effort which is fundamental to the E&V task is the National Bureau of Standards' Taxonomy for an APSE [3], which systematically defines tool capabilities for a full APSE. A third effort, at the Air Force Wright Aeronautical Laboratories [4], provided an initial APSE evaluation questionnaire that can be used as a baseline from which to develop a more refined, comprehensive, and generic set of questions. Finally, previous and current efforts, sponsored by the AJPO, at Virginia Polytechnic Institute and State University [5] have addressed issues associated with validation in APSEs.

In June 1983 the AJPO proposed the formation of the E&V Task and a tri-service APSE E&V Team, with the Air Force designated as lead service [6]. In October 1983 the Air Force officially accepted responsibility as lead service on the E&V Task [7]. The purpose of the E&V Team is to develop the techniques and tools which will provide a capability to perform assessment of APSEs and to determine conformance of APSEs to the CAIS. As the E&V technology is developed, it will be made available to the community for use by DoD components, industry, and academia as deemed appropriate by the respective organizations. The E&V Task will not provide an E&V Organization which will be responsible for the execution of E&V procedures on all APSEs.

2. SCOPE

The overall goal of the E&V Task is to develop and provide to the community the technology for the Evaluation & Validation of APSEs. The E&V Task, sponsored by the AJPO, will be accomplished by an E&V Team which consists of representatives from the Air Force, Army, Navy and other selected agencies. The Air Force has assumed responsibility as lead service for this effort and the Air Force Wright Aeronautical Laboratories (AFWAL) has assumed responsibility as the lead Air Force organization.

In order to accomplish the goal of the E&V Task, eleven specific objectives have been identified. Note that each objective is preceded by "O-" (indicating Objective) and a unique number. This nomenclature is provided to enable illustration of a direct mapping of the E&V Work Breakdown Structure elements (provided in Section 7) to the following specific objectives:

- O-1: Develop Requirements for APSE E&V

    * As a prerequisite to the development of APSE E&V technology, E&V requirements must be specified. The development of E&V requirements will be based upon examination of APSE related issues such as life-cycle methodologies, human engineering aspects, software engineering practices, etc. The E&V requirements which are developed will be used to guide the E&V technical effort.

- O-2: Develop APSE E&V Classification Schema

    * The technical approach to classifying APSE components will be based upon an APSE E&V Classification Schema. This schema is comprised of three major factors: (1) identification of APSE components; (2) identification of associated APSE interface areas for each APSE component; and (3) identification of the appropriate evaluation or validation capability associated with each APSE component. Section 3 (E&V TECHNICAL APPROACH) of this document provides additional detail on the APSE E&V Classification Schema which will be used

initially by the E&V Task. This schema will be refined during the E&V Task.

- O-3: Identify and Classify APSE Components

  * APSE components will be identified and classified based upon the existence of criteria and standards as well as the existence of metrics capabilities for those components. The identification and classification of APSE components will be in accordance with the APSE E&V Classification Schema.

- O-4: Develop APSE Evaluation Capability

  * An evaluation capability will be developed for all APSE components for which there exist no formal standards (i.e., MIL-STD, ANSI, etc.). The evaluation capability for some components will be provided through established metrics, whereas the evaluation capability for other components may be limited to a detailed questionnaire.

  * As a first step toward achieving this objective, previous AFWAL efforts in the area of APSE evaluation will be reviewed for applicability as a baseline for generic evaluation criteria. Because evaluation criteria will be largely dependent upon the defined functionality of each tool, an analysis will be made of the functionality of various tools provided in the DoD APSEs to determine commonality among tool names and tool functions. This analysis will be closely coordinated with the National Bureau of Standards (NBS) effort in defining a taxonomy of APSE tool features. Ongoing standards development activities will be reviewed as a potential source of evaluation criteria and public presentation of the findings of the analysis will be used to solicit input from industry and academia so as to generate a sound and realistic expansion of the developed criteria.

- O-5: Develop APSE Validation Capability

  * A validation capability will be developed for the CAIS, which is currently being developed by the KIT/KITIA, and which will become a MIL-STD. If other APSE related standards are established (i.e., possibly DIANA) appropriate validation capabilities will be developed. Examination of the current validation procedures and Ada Compiler Validation Capability (ACVC) test suite utilized by the Ada Validation Organization (AVO), as well as procedures implemented by ANSI and ISO, will be used as a baseline. The APSE validation studies performed by Virginia Polytechnic Institute and State University, and the current Formal Qualification Tests (FQT) being applied to the ALS KAPSE (and those which must be similarly developed for the AIE KAPSE) provide an available baseline from which a validation capability may be developed.

- O-6: Monitor the Formal Qualification Testing (FQT) of DoD APSEs

  * The development of the Army's ALS has progressed to the stage of FQT, and the Army is currently utilizing a tri-service team to assist in the monitoring of the ALS testing. Such tri-service testing cooperation is also envisioned by the Air Force for its testing of the AIE and the Navy for its ALS/N. The FQT test suite developed by each service will provide a useful baseline for examining the various tool/user/interface aspects of an APSE, and a realistic approach to determining the capabilities of the DoD APSEs.

- O-7: Develop Evaluation & Validation Tools and Aids

  * As the requirements for E&V are determined, various software tools/aids will be identified as essential to the E&V effort. Such tools/aids

include test sets, test scenarios, data reduction capability, and other designated means of automated support. As these tools/aids become more clearly defined, an assessment will be made to include such capability. Existing tools/aids which are applicable to the E&V Task will be considered for use. New tools/aids which are determined to be essential for the APSE E&V Task will be assessed for possible contractor development. One specific validation capability which will be developed through a contractual effort will be the CAIS Validation Capability (CVC). The existing Ada Compiler Validation Capability (ACVC) will be included as part of the E&V Tools/Aids.

- O-8: Develop Procedures for Implementation of E&V

  * The E&V Task will develop and provide the technology and procedures by which E&V of APSEs will be accomplished. It will not provide an E&V Organization which will be responsible for the execution of evaluation and validation procedures on all APSEs. The E&V procedures will be based upon E&V requirements, APSE standards, evaluation criteria, validation capability, and existing E&V tools/aids. Once the procedures and mechanisms are fully developed, the APSE Validation execution responsibility will be transitioned to an appropriate validation organization. The APSE Evaluation capability will be transitioned to the community for use by DoD components, industry, and academia.

- O-9: Provide Initiative and Focal Point With Respect to APSE E&V

  * There currently exists a need to provide a focal point for APSE developers and users with regard to information about E&V of APSEs. APSE E&V questions arise frequently within professional societies and user groups. A forum is needed in

which APSE E&V questions can be addressed and discussed, and in which APSE E&V information can be disseminated throughout the Ada community.

* The E&V Team will provide a focal point for APSE E&V for the Ada community. Public reports on the results of this APSE E&V Plan will be made available to professional societies such as AdaTEC and AdaJUG. This is in keeping with the AJPO philosphy of public exposure. The E&V task is the lead DoD effort with regard to APSE E&V. In this respect, the E&V Team will participate in, and assist where possible, other programs connected with APSE E&V. Such programs include the KIT/KITIA, METHODMAN, and international development efforts.

- 0-10: Solicit Industry/Academia Participation in the E&V Task

  * In order to encourage industry/academia participation in the E&V effort, an E&V Workshop will be conducted on an annual basis throughout the duration of the E&V Task. Information on the E&V Workshops will be made publically available and participants will be selected on the basis of position papers which are written relevant to the technical aspects of the E&V Task.

- 0-11: Promote Community Use and Acceptance of the E&V Effort

  * Use of the E&V technology developed through this task will provide for an orderly progression of technology insertion into environments. The E&V technology thus developed will be extendable to other software development efforts, thereby maximizing the economic benefits of the E&V task products and minimizing the cost within DoD and industry of doing E&V related work.

3. E&V TECHNICAL APPROACH

3.1 APSE Concept

The APSE, as depicted by the STONEMAN document, provides a virtual interface between the user of the APSE and the particular host system upon which the APSE is installed. This interface is designed to be machine and operating system independent; in effect, it defines an Ada virtual machine whose features are available on all actual host machines. The purpose of the APSE is to provide an environment for the design, development, documentation, testing, management, and maintenance of embedded computer software, written principally in the Ada programmming language.

The initial efforts of the E&V Task are based upon the concept of an APSE structure as defined by the original STONEMAN document. STONEMAN paints a broad picture of the needs and identifies the relationships of the parts of an integrated APSE. Allowance will be made for the possible modification of that APSE structure based upon a future revised STONEMAN document.

3.2 APSE E&V Classification Schema

The technical approach to the E&V effort requires that APSE components be identified and classified based upon a well-defined Classification Schema. The schema which is initially proposed in this E&V Plan consists of three basic steps:

- Step 1: Identification of APSE Components;
- Step 2: Identification of APSE Interface Classes; and
- Step 3: Identification of APSE E&V Categories.

The following sections present additional detail on each of these steps, as well as an illustration of the result of each step. The E&V Classification Schema which is presented in this document is expected to be further refined during the E&V Task.

3.2.1 Step 1: Identification of APSE Components

For the purpose of the E&V Classification Schema, APSE components are defined to be features of the APSE. The National Bureau of Standards Taxonomy of Tool Features for the APSE [3] presents a hierarchical arrangement of software tool features. The first (highest) level is an abstract level which encompasses all of the features below it. The second level includes the basic processes of the APSE (i.e., input, output, and function). The third level includes the classes of tool features (i.e., subject, control, transformation, static analysis, dynamic analysis, management, user output, and machine output). The fourth and fifth levels include specific APSE features.

Initially, as a basis for Step 1, the National Bureau of Standards Taxonomy will be used to identify APSE components. However, as additional E&V Requirements are specified during the E&V Task, the list of APSE components will be expanded to reflect: (1) additional APSE features; and (2) finer granularity of previously identified APSE features.

This first step of the Classification Schema results in a hierarchical structure which can be illustrated by a list of APSE components, identified through an appropriate numbering scheme. Figure 3-1 illustrates the format for the list of APSE components which result from Step 1.

Component 1

Component 1.1

Component 1.1.1

Component 1.1.2

Component 1.2

Component 1.2.1

Component 1.2.2

Component 1.2.2.1

.
.
.

Figure A-3-1. STEP 1 - IDENTIFICATION OF APSE COMPONENTS

3.2.2 Step 2: Identification of APSE Interface Classes

Following the identification of APSE components, the particular APSE interface classes which are associated with each APSE component must be identified. At present, four classes of APSE interfaces have been identified as being applicable to APSE components. These four classes of interfaces are the Common APSE Interface Set (CAIS), Ada Packages, Inter-tool Data Interfaces, and the User/APSE Interface. These classes of APSE interfaces may be further refined during the E&V Task.

- CLASS 1 Interfaces: (CAIS)

  * The CAIS is being developed by the KIT/KITIA as the foundation for Interoperability and Transportability (I&T) of data and tools. The CAIS provides the virtual operating system on which the tools run, as well as the minimum set of command, edit and similar functions needed to move tools from one CAIS to another. Tools written in Ada, using only the minimum guaranteed characteristics of the CAIS packages, will be portable to all APSEs providing conforming implementations for the CAIS packages used by the tool.

  * The E&V Task will check implementation conformance and address the two way communication across this interface.

  * Examples of CLASS 1 CAIS functionality are Input/Output, Data Management, and Process Management.

- CLASS 2 Interfaces: (Ada Packages)

  * Certain Ada Packages will be established as common or a standard over time that are not part of the CAIS. These packages may interface directly with the CAIS or may use other standard packages in their implementation. Each package can then be considered an additional layer of abstraction between the CAIS and the User.

* The E&V Task will address the two way communication across both the external visible part and the implementation interface. As part of the implementation portion, the E&V task will check to ensure that the package meets the various aspects of Interoperability and Transportability as developed by the KIT/KITIA.

* Examples of CLASS 2 packages would be packages provided in a Math Package Library.

- CLASS 3 Interfaces: (Inter-tool Data Interfaces)

  * As sets of tools are identified which use similar data, inter-tool data interface commonality will be established.

  * The E&V Task will address the inter-tool data interfaces, with respect to the data types and the operations on that data.

  * Examples of CLASS 3 Inter-tool Data interfaces are DIANA and, potentially, any file format (data types and operations on those files); and such information as performance and statistical data concerning the operation and performance of the APSE and individual tools.

- CLASS 4 Interfaces: (User/APSE Interfaces)

  * The user looks at the APSE from the outside and sees particular tool interfaces as well as an APSE SYSTEM.

  * The E&V Task will address both the user/tool interface and the user's SYSTEM view of the APSE.

  * Examples of CLASS 4 interfaces would be "Control C" always having the same meaning, or all editors using the same subset of functions, or the same data naming structure or Command Language. Anything that the user sees from the time of

log-on to log-off, including performance, would fall into this category. CLASS 4 would include overall completeness of an APSE in terms of its functionality as well as the ease of transition from one life-cycle phase oriented tool set to another.

As the second step in the E&V Classification Schema, each APSE component will be examined to determine which APSE interface classes are affected by each component. Figure 3-2 illustrates the format for the 2-dimensional matrix which results from Step 2 of the E&V Classification Schema.

APSE COMPONENTS	APSE INTERFACE CLASSES			
	1	2	3	4
Component 1				
Component 1.1				
Component 1.1.1				
Component 1.1.2				
Component 1.2				
Component 1.2.1				
Component 1.2.2				
Component 1.2.2.1				
. . .				

Figure A-3-2. STEP 2 - IDENTIFICATION OF APSE INTERFACE CLASSES

3.2.3 Step 3: Identification of APSE E&V Categories

For the purpose of the E&V Classification Schema, the term "Evaluation" represents a method of assessing the quality of APSE components for which no specific standard (i.e., MIL-STD, ANSI, etc.) exists, or for which a standard may exist but there is no known capability to measure conformance to that standard. The term "Validation" represents a method of determining conformance to a standard which is applicable to an APSE (e.g., MIL-STD-1815A, CAIS, etc.).

The determination of what methodology (i.e., evaluation or validation) is then based on whether a standard exists and whether a means of checking conformance to that standard also exists. Different levels of conformance checking exist and that leads to a paritioning of validation methodology into non-formal and formal techniques. Based on this notion of standards and conformance checking, the following categories are provided for determining appropriate assessment methodology.

- Category A:

  * If no standard for an APSE component exists and no technique of evaluating conformance has been developed, then the component requires subjective evaluation.

- Category B:

  * If no standard for an APSE component exists, but a method for assessing the quality (i.e., a metrics capability) exists, then the component requires objective evaluation.

- Category C:

  * If a standard for an APSE component exists but there is no existing method for determining conformance to that standard, then the component is in an intermediate category.

- Category D:

  * If both a standard for an APSE component and a method for determining conformance to that standard exist, then the component requires validation.

- Category E:

  * If a standard for an APSE component and a purely formal technique for determining conformance to that standard exist, then the component requires formal validation.

When these categories are applied to APSE components the appropriate quality assessment technology for each component type may be easily determined.

As the third step in the E&V Classification Schema, each APSE component/APSE Interface Class couple will be examined to determine which APSE E&V Category is most appropriate, based upon existing standards/criteria and metrics capabilities. Figure 3-3 illustrates the format for the 3-dimensional matrix which results from Step 3 of the E&V Classification Schema.

The result of categorizing APSE components into the appropriate APSE interface areas and E&V categories is primarily to determine what standards and assessment techniques have to be developed in an independent manner. In other words, the E&V Classification Schema allows the decision to pursue the development of standards, validation methods, or formal methods independently of what course may be chosen for other components even in the same area of application.

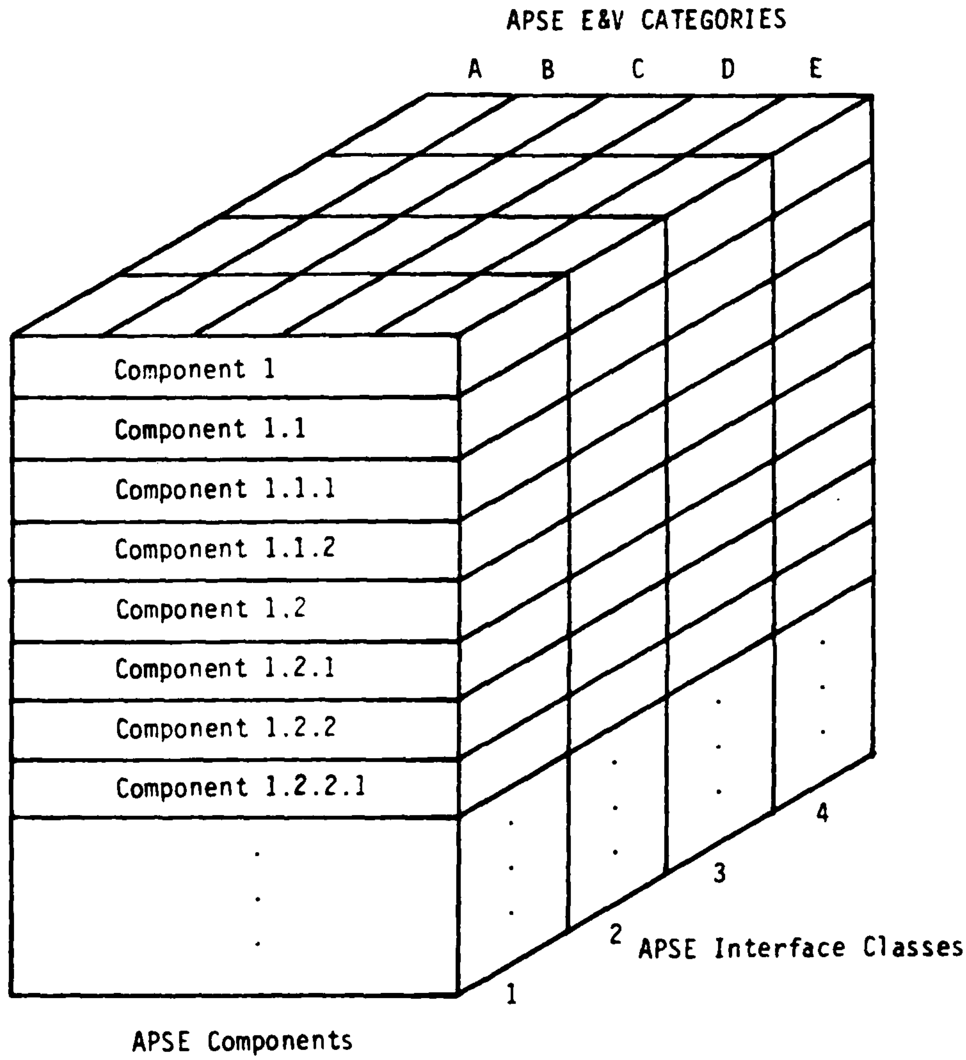


Figure A-3-3. STEP 3 - IDENTIFICATION OF APSE E&V CATEGORIES

4. E&V MANAGEMENT APPROACH

Figure 4-1 depicts the E&V management structure. Each of the components is identified in the following sections.

4.1 Ada Joint Program Office

The Ada Joint Program Office (AJPO) sponsors the E&V task. All E&V Team activities are coordinated with the AJPO through the E&V Team Chairperson. The AJPO requires that the status of the E&V task be briefed to the AJPO, as well as to the three service representatives, at quarterly Ada tri-service reviews.

4.2 Air Force, Army, Navy

The Air Force has assumed responsibility as lead service on the tri-service E&V Task. However, the status of the E&V Task is briefed to the AJPO and the service representatives at quarterly Ada tri-service reviews. At these reviews, each service representative has the opportunity to request additional information on the E&V Task and to recommend modifications to the proposed E&V Task planning.

4.3 E&V Team Chairperson

The Air Force Wright Areonautical Laboratories (AFWAL) has assumed responsibility as the lead Air Force organization for the E&V Task. The E&V Team Chairperson is an AFWAL representative who is authorized to work directly with the AJPO in the execution of the E&V Task. The E&V Team Chairperson is required to brief the status of the E&V Task to the AJPO and services at quarterly Ada tri-service reviews. The E&V Team Chairperson is fully responsible for providing technical direction to the E&V Team members and for coordinating all of the E&V activities.

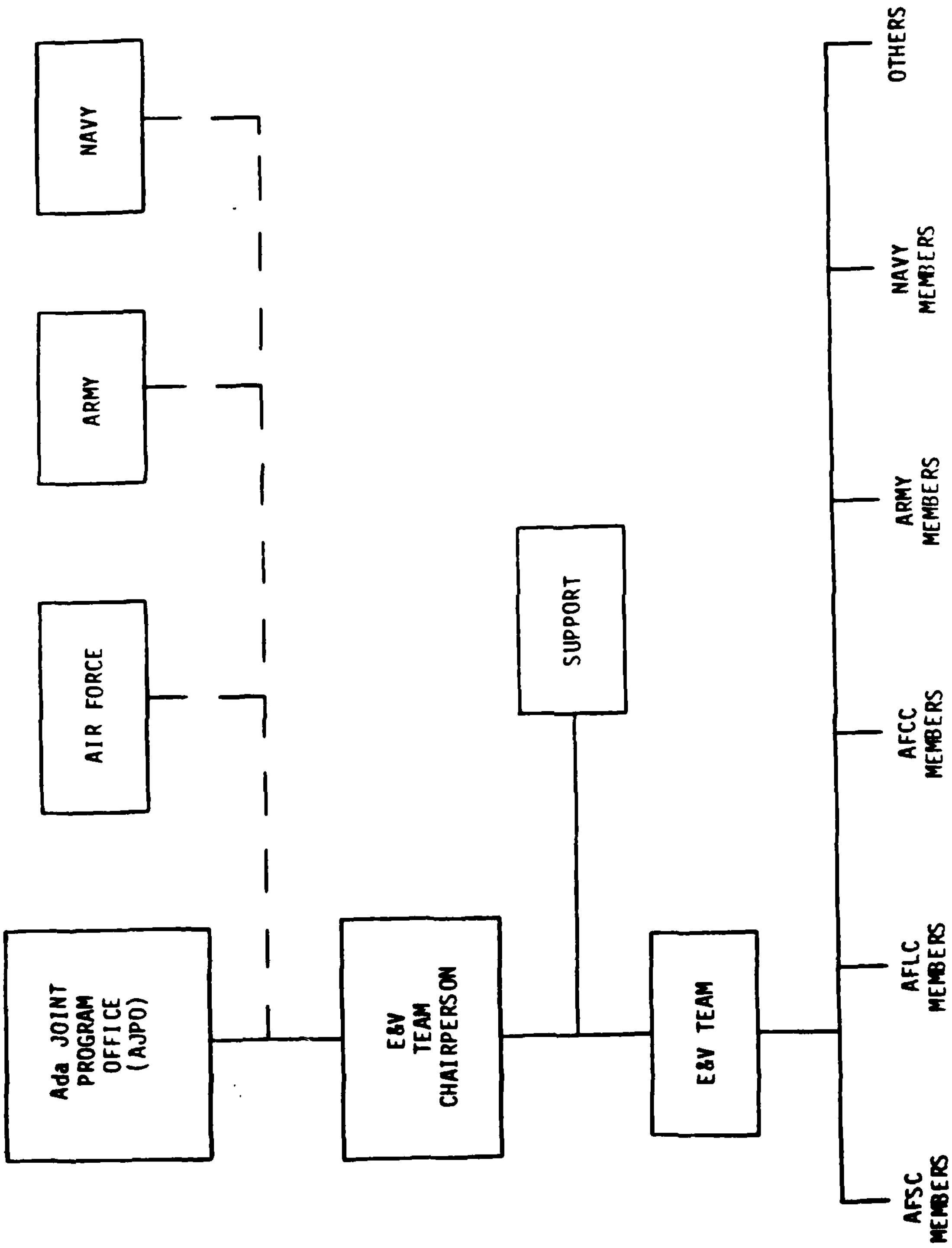


Figure A-4-1. E&V MANAGEMENT STRUCTURE

4.4 E&V Team

The E&V Team is composed of representatives from the following organizations:

- Air Force

  * Air Force Systems Command

  * Air Force Logistics Command

  * Air Force Communications Command

- Army

- Navy

- Other Selected Agencies

E&V Team meetings are convened quarterly and E&V Team members are responsible for representing the technical issues/concerns of their respective organizations at these meetings. Similarly, E&V Team members are responsible for reporting the status of the E&V Team activities to their repective organizations.

4.5 E&V Team Working Groups

In order to coordinate all of the activites to be accomplished within the E&V Task, the E&V Team will be partitioned into five working groups. The identification of these working groups, and their associated areas of responsibility, are delineated in the following sections. These working groups are subject to change during the life of the E&V Task. Each working group will have a designated Chairperson and Vice-Chairperson. It will be the responsibility of each working group Chairperson to coordinate the activities of the working group with the E&V Team Chairperson. In addition, each working group Chairperson will be required to brief the status of the respective working group at every E&V Team meeting.

4.5.1 E&V Requirements Working Group (REQWG)

The REQWG shall be responsible for performing the following tasks:

- Review life-cycle methodology materials and determine life-cycle issues which should be addressed by the E&V task.

- Develop an E&V Requirements Document against which the E&V Reference Manual will be developed.

- Refine E&V Requirements outputs from the E&V Workshop.

- Provide analysis of E&V Requirements to determine completeness, traceability, testability, consistency and feasibility.

- Identify issues which may impact the development of E&V technology but which do not necessarily correlate to APSE components.

- Provide recommendations for development/acquisition of E&V tools/aids through the development of an E&V Tools/Aids Requirements Document.

- Prepare position papers through the duration of the E&V Task which address issues on E&V Requirements.

4.5.2 E&V Technical Coordination Working Group (TECWG)

The TECWG shall be responsible for performing the following tasks:

- Perform literature search for efforts relevant to the E&V task.

- Develop a Technical Coordination Strategy Document which will:

    * identify related technical efforts;

* identify relationships between the E&V Task and each of the related tasks;

* identify areas of mutual benefit to the tasks;

* identify impact of schedules;

* identify level of coordination required;

* identify issues which require resolution to the mutual benefit of the tasks involved.

- Provide technical presentations to the E&V Team on these related efforts.

- Prepare position papers throughout the duration of the E&V task which address particular aspects of the E&V Task with related tasks/efforts.

4.5.3 E&V APSE Analysts Working Group (APSEWG)

The APSEWG shall be responsible for performing the following tasks:

- Obtain expertise on DoD and commercial APSEs available within the DoD.

- Prepare presentations to the E&V Team on these APSEs, each APSE presentation increasing in level of detail.

- Identify existing capabilities/tests/tools associated with each APSE which will assist the E&V effort.

- Develop evaluation criteria to be applied to existing DoD APSEs.

- Provide analysis of application of evaluation criteria to DoD APSEs.

- Monitor the DoD APSE Formal Qualification Testing.

- Prepare position papers throughout the duration of the E&V Task which address particular aspects of the DoD

APSEs in relation to the E&V effort.

4.5.4 E&V Common APSE Interface Set Working Group (CAISWG)

The CAISWG shall be responsible for performing the following tasks:

- Emphasize study on the CAIS.
- Review the development of the CAIS and identify areas of possible concern to E&V.
- Monitor the DoD APSE Formal Qualification Testing.
- Provide presentations to the E&V Team on the CAIS.
- Provide liaison activity to the KIT/KITIA.
- Review existing DoD KAPSE interface tests and augment to provide initial test set for CAIS MIL-STD Version 1.
- Recommend specific areas of consideration for the CAIS Validation Capability Statement of Work.
- Develop a Validation Procedures Document which will provide details on the validation procedures to be implemented by organizations to which the CAIS validation responsibility will be transferred.
- Prepare position papers throughout the duration of the E&V Task which address particular aspects of the CAIS as relevant to E&V.

4.5.5 E&V Public Coordination Working Group (PUBWG)

The PUBWG shall be responsible for performing the following tasks:

- Identify professional organizations which are technically related to the E&V effort.
- Develop a Public Coordination Strategy Document which

provides an approach as to how such public coordination will be performed.

- Record minutes of all E&V Team meetings.

- Coordinate the various E&V papers/documents which shall be included in the annual E&V Public Report.

- Prepare a set of E&V viewgraphs and corresponding text to allow E&V Team members to present the status of the E&V Task at public meetings.

- Prepare E&V status reports for publication in related journals and newsletters.

- Catalog all issues related to the E&V effort.

- Develop and maintain an E&V project reference list.

4.6 Support

Contractor support for the E&V task will be obtained for the purpose of development, publication, distribution, and configuration management of all APSE E&V documentation. Additional contractor support will be obtained for the purpose of developing software tools/aids to be used for evaluation and validation of APSEs. Such support will include development of a CAIS Validation Capability (CVC) which will be used to determine conformance of an APSE to the CAIS, which is currently under development by the KIT/KITIA.

5. E&V RELATIONSHIP TO OTHER ORGANIZATIONS

Figure 5-1 illustrates the relationship of the E&V Task to other organizations.

5.1 Air Force, Army, Navy APSE Development Efforts

The Army and Air Force have begun separate developments of an APSE; the Navy intends to procure a development which will be based largely on the Army's APSE, but which will be tailored to meet specific Navy requirements. The contractor for the Army's ALS is SofTech, Inc.; the contractor for the Air Force's AIE is Intermetrics, Inc.; and the contractor for the Navy's ALS/N is yet to be determined. The E&V Team will interact with the three services and their respective APSE contractors for information exchange and consultation, particularly in the area of Formal Qualification Testing (FQT).

5.2 KIT/KITIA

The purpose of the KIT and KITIA, under the direction of the AJPO, is to develop a standard set of KAPSE interfaces to ensure the transportability of tools and the interoperability of data between conforming APSEs. The E&V Team will interact with the KIT and KITIA for information exchange, particularly in the area of APSE interfaces, and for initial review of E&V work prior to public exposure. Several members of the E&V Team are also members of the KIT/KITIA. The Chairperson of the KIT is also a guest member of the E&V Team.

5.3 METHODMAN

The purpose of the METHODMAN effort, under the direction of the AJPO, is to develop requirements and encourage the development of methodologies to support the entire software development life-cycle. One of the goals of the METHODMAN effort will be the construction of a complete set of tools to support a selected methodology. The E&V Team will interact with the METHODMAN effort for information exchange, particularly in the areas of tool definitions, evaluations, and validation. The Chairperson of the METHODMAN effort is also a guest member of the E&V Team.

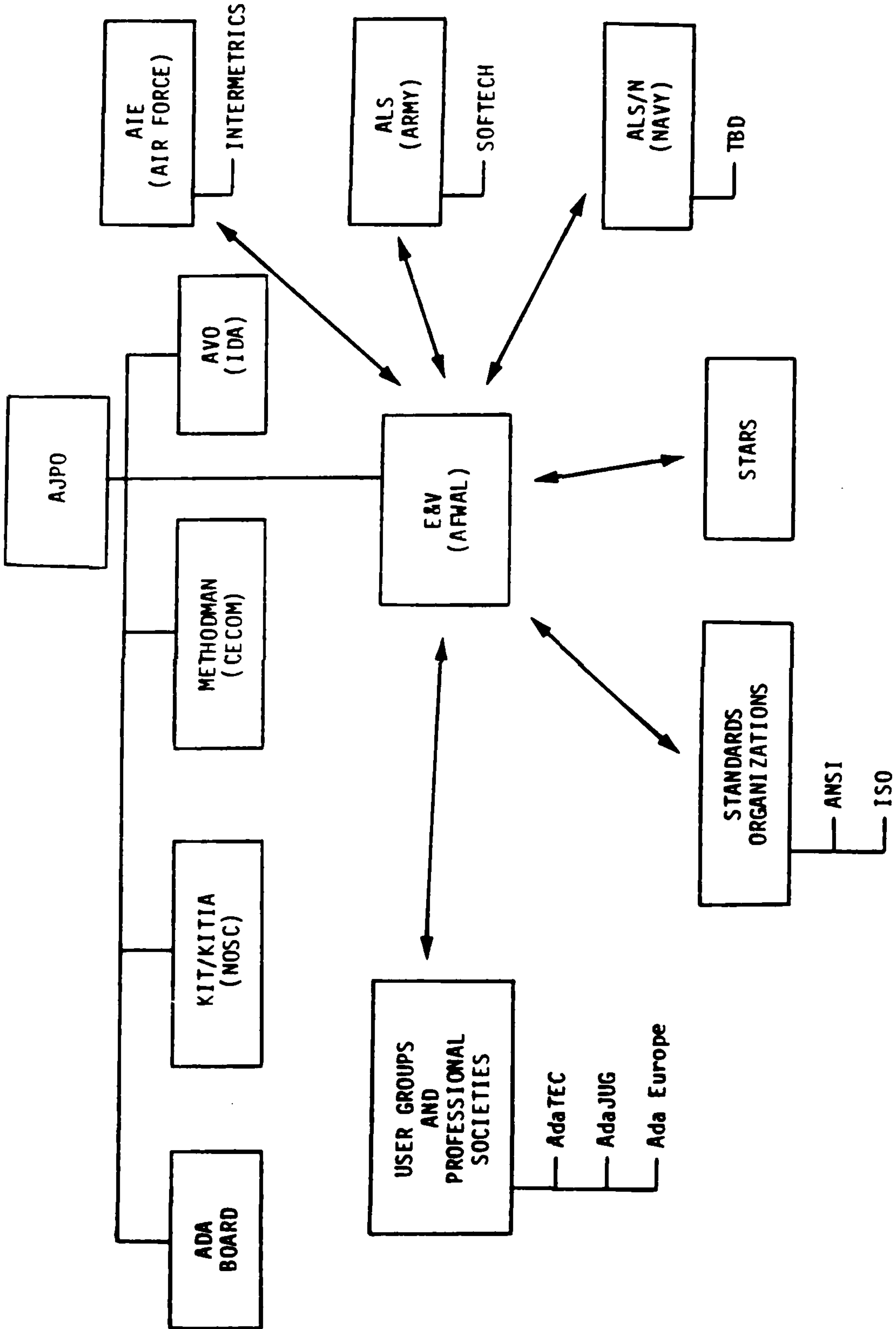


Figure A-5-1. E&V RELATIONSHIP TO OTHER ORGANIZATIONS

5.4 Ada Validation Organization (AVO)

The purpose of the AVO, under the direction of the AJPO, is to ensure that compiler developers have correctly implemented the standard Ada language (ANSI/MIL-STD-1815A-1983). The AVO has established formal procedures for validating Ada compilers and mechanisms by which the validation procedures are executed. The E&V Team will interact with the AVO for information exchange, particularly in the area of validation requirements and procedures. The compiler validation capability will be incorporated within the E&V technology developed by the E&V Team.

5.5 User Groups and Professional Societies

It is anticipated that AdaTEC, the Ada-JOVIAL Users Group (AdaJUG), and Ada Europe will provide valuable contributions to the APSE E&V effort. The E&V Team has no formal relationship with these groups; however, the E&V Team will use some or all of these groups as regular forums for the presentation of reports and technical results of the APSE E&V effort, and will solicit inputs from members.

5.6 Standards Organizations

As with the User Groups and Professional Societies, there is no formal relationship with the Standards Organizations. However, because the E&V Task is based upon validation of KIT/KITIA developed standards, the E&V Team must be familiar with the procedures for enforcement of standards. Knowledge of how standards are currently enforced will provide useful guidelines for the direction of the E&V Task.

5.7 AJPO Director's Advisory (ADA) Board

The purpose of the ADA Board is to advise the director of the AJPO with regard to policy and issues related to the Ada Program. The E&V Team will interact with the ADA Board for information exchange on issues related to the APSE E&V effort.

5.8 Software Technology for Adaptable, Reliable Systems (STARS)

The STARS Program, under the direction of the DoD, was established to develop and promote new software technology. The E&V effort will be closely coordinated with the STARS tasks. E&V Team liaisons will be provided to the STARS Program to ensure coordination of technical objectives as well as scheduled milestones.

6. E&V DELIVERABLES

This section delineates each of the deliverables of the E&V Task. Working as a whole, the E&V Team members, the technical consultants, and the technical support contractors, are responsible for the development of all documents. However, in order to more clearly reflect the areas of emphasis for the E&V working groups and support personnel, each document description specifies the individuals who are primarily responsible for the development of that document.

- E&V Plan

  * The E&V Plan provides a detailed and organized approach to the accomplishment of the E&V Task. The E&V Plan reflects the management approach, the technical approach and the schedules for all E&V activities. The E&V Plan is considered to be evolutionary and will be updated on an annual basis to reflect possible proposed modifications to the approach and/or schedules and to reflect accommplishments during the previous year. The E&V Team Chairperson is primarily responsible for the development of the E&V Plan.

- E&V Public Report

  * The E&V Public Report, which will be made available to the public on an annual basis, will provide information on the acitivities of the E&V Team. The E&V Public Report will contain the recorded minutes of all E&V Team meetings as well as all position papers prepared by E&V Team members. The E&V Public Report will also contain E&V position papers written by industry/academia participants in the annual E&V Workshop, as well as all documentation which results from the E&V Workshop. The Public Coordination Working Group (PUBWG) is primarily responsible for the format and collation of all entries in the E&V Public Report.

- E&V Project Reference List

    * The E&V Project Reference List will provide a list of documents used as reference material by the E&V Team. Corresponding with each specified document will be a snyopsis which identifies the relevance of that document to the E&V Task. The E&V Project Reference List will be expanded throughout the duration of the E&V Task. The Public Coordination Working Group (PUBWG) is primarily responsible for the development of the E&V Project Reference List.

- E&V Technical Coordination Strategy Document

    * The E&V Technical Coordination Strategy Document will identify other ongoing DoD/contractual efforts which are technically related to the E&V Task. This document will provide a strategy for coordination between the E&V Task and each identified effort. It will specify level of coordination, points of contact, impact of schedules of one effort on another, and benefits to be gained by each effort as a result of such coordination. This document will be updated throughout the duration of the E&V Task in order to incorporate efforts which are initiated during this time. The Technical Coordination Working Group (TECWG) is primarily responsible for the development of the E&V Technical Coordination Strategy Document.

- E&V Public Coordination Strategy Document

    * The E&V Public Coordination Strategy Document will identify public organizations/activities with which coordination should be established with the E&V Task for the benefit of information exchange. This document will provide a strategy for coordination between the E&V Task and each of these organizations/activities. It will specify level of coordination, points of contact, and

procedures by which the plans and accomplishments of the E&V Task are presented to the organizations/activities. This document will be updated throughout the duration of the E&V Task in order to incorporate organizations/activities which are initiated during this time. The Public Coordination Working Group (PUBWG) is primarily responsible for the development of the E&V Public Coordination Strategy Document.

- E&V Requirements Document

  * The E&V Requirements Document will identify the requirements by which the E&V technology will be developed. E&V requirements will be based upon review of life-cycle methodologies, software engineering practices, human engineering aspects associated with software development, and other issues relevant to APSEs. The Requirements Working Group (REQWG) will be primarily responsible for the development of the E&V Requirements Document.

- DoD APSE Analysis

  * The DoD APSE Analysis will provide information on the features provided in the DoD APSEs. This analysis will reflect areas of commonality as well as areas of discrepancy in the manner in which functions are performed. Each revision of the DoD APSE Analysis will provide additional detail on the comparative analysis. The APSE Working Group (APSEWG) is primarily responsible for the development of the DoD APSE Analysis.

- APSE Validation Procedures Document

  * The APSE Validation Procedures Document will provide details on the validation procedures to be implemented by organizations to which the validation execution responsibility will be

transferred. Initial versions of the APSE Validation Procedures Document will reflect general validation procedures common to existing validation organizations. Later versions of the APSE Validation Procedures Document will include APSE specific validation procedures, such as those applicable to the CAIS. The CAIS Working Group (CAISWG) is primarily responsible for the development of the APSE Validation Procedures Document.

- E&V Configuration Management Plan

  * The E&V Configuration Mangement Plan will specify the procedures which must be followed in order to perform Configuration Management of all E&V documents generated by the E&V Task as well as all tools/aids developed by the E&V Task. The Configuration Manaagement Plan will be consistent with current Configuration Management policies implemented by the Avionics Laboratory at Wright-Patterson Air Force Base. The E&V Technical Support Contractor is primarily responsible for the development of the E&V Configuration Management Plan.

- E&V Classification Schema Document

  * The E&V Classification Schema Document will be used to define the approach for classification of APSE components. The initial E&V Classification Schema is provided in Section 3 (TECHNICAL APPROACH). However, as the E&V Task begins to identify and classify APSE components, the initial schema will be refined. The E&V Technical Support Contractor is primarily responsible for the development of the E&V Classification Schema Document.

- E&V Reference Manual

* The E&V Reference Manual will provide information on the classification of APSE components. For each identified APSE component, the E&V Reference Manual will identify the corresponding criterion/standard associated with that APSE component, as well as the metrics capability (or questionnaire entries) which are used to access that APSE component. Throughout the E&V Task, the E&V Reference Manual will be expanded to reflect finer granularity in the identification of APSE components as well as newly acquired/developed metrics capabilities. The E&V Technical Support Contractor is primarily responsible for the development of the E&V Reference Manual.

- E&V Guidebook

  * The E&V Guidebook is a companion document o the E&V Reference Manual. It provides information to the user as to how to implement the tools/techniques identified in the E&V Reference Manual for appropriate application of the E&V technology. The E&V Technical Support Contractor is primarily responsible for the development of the E&V Guidebook.

- E&V Tools/Aids Requirements Document

  * The E&V Tools/Aids Requirements Document will specify the requirements to be used for the selection of E&V tools/aids to be acquired/developed as part of the E&V Task. This document will also include the rationale for establishing priorities for the acquisition/development of such tools/aids. The Requirements Working Group (REQWG) is primarily responsible for the development of the E&V Tools/Aids Requirements Document

- E&V Tools/Aids

* As the E&V Task Progresses, APSE components will be classified and existing E&V capabilities for those components will be identified. The E&V Tools/Aids Requirements Document developed by the REQWG will be used to specify and prioritorize E&V tools/aids which must be developed. Based upon the E&V Tools/Aids Requirements Document, additional contractual efforts will be initiated for the development of such E&V tools/aids.

- CAIS Validation Capability (CVC)

  * The CAIS Validation Capability (CVC) will provide the validation capability to determine APSE conformance to the CAIS as specified in the future CAIS MIL-STD. Initial efforts to provide such a capability will consist of review and adaptation of existing DoD KAPSE interface Formal Qualification Tests by the CAIS Working Group (CAISWG). However, a separate contractual effort will be used to develop a full validation capability.

7. E&V WORK BREAKDOWN STRUCTURE

Figure 7-1 depicts the areas of E&V Task responsibility which include the following:

- APSE E&V Management
- APSE E&V Requirements
- APSE E&V Reference Manual Development
- APSE Evaluation Capability
- APSE Validation Capability
- APSE E&V Tools/Aids
- APSE E&V Support

A Work Breakdown Structure (WBS) is provided for each of the above areas of responsibility.

Figure 7-2 illustrates the relationship of each WBS element to the specific objectives identified in Section 2 of this document.

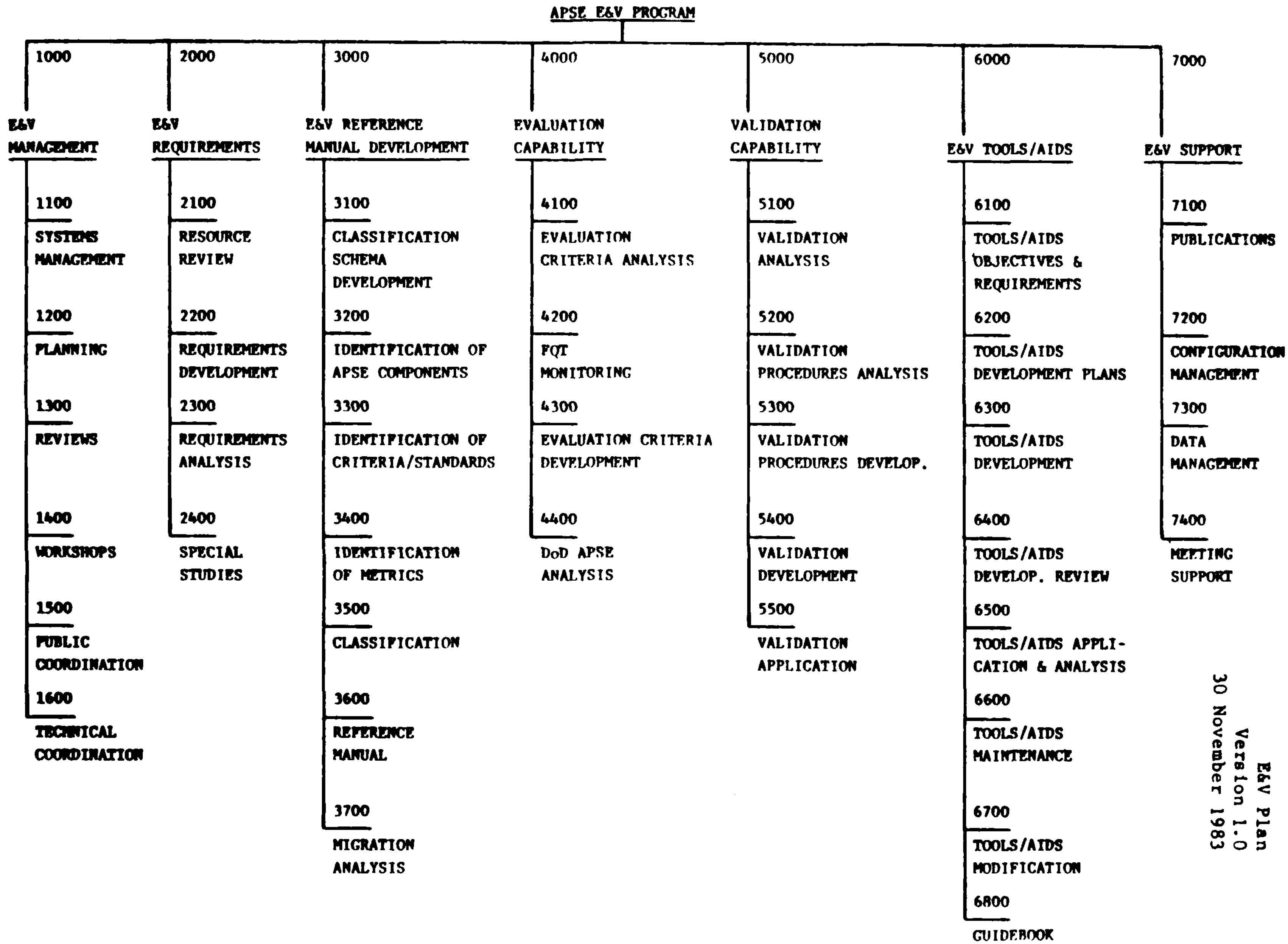


Figure A-7-1. APSE E&V TASK WORK BREAKDOWN STRUCTURE

WBS ELEMENT \ E&V OBJECTIVE		0-1	0-2	0-3	0-4	0-5	0-6	0-7	0-8	0-9	0-10	0-11
1000:	1100									X		
	1200									X		
	1300									X		
	1400										X	
	1500											X
	1600											X
2000:	2100	X			X	X			X			
	2200	X										
	2300	X										
	2400	X										
3000:	3100		X									
	3200			X								
	3300			X								
	3400			X								
	3500			X								
	3600			X								
	3700			X				X				
4000:	4100				X							
	4200				X		X					
	4300				X							
	4400				X							
5000:	5100					X						
	5200								X			
	5300								X			
	5400					X						
	5500					X						
6000:	6100							X				
	6200							X				
	6300					X		X				
	6400							X				
	6500							X				
	6600							X				
	6700							X				
	6800							X	X			
7000:	7100										X	
	7200									X		
	7300									X		
	7400									X		

Figure A-7-2. MAPPING OF WBS ELEMENTS TO E&V OBJECTIVES

7.1 1000 APSE E&V Management

- 1100 APSE E&V Systems Management

  * This WBS element provides for management of the APSE E&V Task. It further provides for a Public Report to be prepared every year. The Public Report will cover the technical accomplishments of the APSE E&V Task for the prior year and will be suitable for distribution in hard copy.

- 1200 APSE E&V Planning

  * This WBS element provides for the planning necessary to follow through and complete the APSE E&V Task. It further provides for the updating of the APSE E&V Plan on an annual basis.

- 1300 APSE E&V Reviews

  * This WBS element provides for the preparation and presentation of the E&V Task progress to the Ada Joint Program Office and the three services at the quarterly Ada tri-service Reviews.

1400 APSE E&V Workshops

  * This WBS element provides for the organization and management of an E&V Workshop, which will be conducted on an annual basis throughout the duration of the E&V Task. The purpose of the E&V Workshop will be to encourage industry/academia participation in the E&V effort. Participation in the E&V Workshop will be limited. Information on the proposed E&V Workshop will be made publically available and participants will be selected on the basis of position papers which are written relevant to the technical aspects of the E&V Task.

- 1500 APSE E&V Public Coordination

  * This WBS provides for the development of a strategy by which the E&V Team will maintain coordination with the public on the progress of the E&V Task. This WBS includes preparation of E&V articles to be submitted for publication. It also includes preparation of materials which may be utilized by the E&V Team members for public presentation on the E&V Task.

- 1600 APSE E&V Technical Coordination

  * This WBS provides for the development of a strategy by which the E&V Team will maintain coordination with other related technical efforts. This WBS includes: (1) the identification of related tasks; (2) the identification of the relationships between the E&V Task and each of the related tasks; (3) the identification of areas of mutual benefit to the tasks; (4) the impact of task schedules; (5) the identification of level of coordination required; and (6) the identification of issues which require resolution to the mutual benefit of the tasks involved.

7.2 2000 APSE E&V Requirements

- 2100 APSE E&V Resource Review

  * This WBS element provides for the review of literature and documentation applicable to APSE E&V requirements. Such literature and documentation will include subjects such as evaluation and validation studies, standards enforcement, tool functionality, APSE requirements, etc.

- 2200 APSE E&V Requirements Development

  * This WBS element provides for the development of requirements for APSE E&V. These requirements will be documented in an E&V Requirements Document which will be revised throughout the duration of the E&V Task as new requirements are identified.

- 2300 APSE E&V Requirements Analysis

  * This WBS element provides for the analysis of APSE E&V Requirements developed under WBS element 2200. This analysis will be conducted to determine completeness, traceability, testability, consistency and feasibility.

- 2400 APSE E&V Special Studies

  * This WBS element provides for any technical analysis or study not mentioned elsewhere. Specifically included are studies resulting in methods for assessing the risk associated with achieving levels of APSE E&V and cost benefit analysis that will provide a quantitative means to assist in making recommendations and decisions concerning implementation.

7.3 3000 APSE E&V Reference Manual Development

- 3100 APSE E&V Classification Schema Development

  * This WBS element provides for the development of a general schema which will be used as a basis for classification of APSE components. This schema will initially be based upon the classification schema provided in Section 3 of this document.

- 3200 Identification of APSE Components

  * This WBS element provides for the identification of APSE components, based upon the functionality and interface areas presented in Section 3 of this document.

- 3300 Identification of Criteria/Standards for APSE Components

  * This WBS element provides for the identification of existing criteria or standards for each of the APSE components identified under WBS 3200. If no criteria or standards exist for a particular APSE component, then this WBS will result in recommendations for the development of criteria against which that component may be evaluated.

- 3400 Identification of Metrics for Criteria/Standards

  * This WBS element provides for the identification of existing metrics for the criteria/standards identified under WBS 3300. If no metrics exist for a particular criterion or standard, then this WBS will result in recommendations for the development of metrics associated with that criterion or standard.

- 3500 E&V Classification

  * This WBS element provides for the classification of all APSE components identified under WBS 3200, based upon the schema developed under WBS 3100 and the associated criteria/standards and metrics identified under WBS 3300 and WBS 3400, respectively.

- 3600 E&V Reference Manual

  * This WBS element provides for the documentation of the results obtained in WBS 3500 in an E&V Reference Manual.

- 3700 APSE E&V Migration Analysis

  * This WBS element provides for a continuing analysis of the results obtained under WBS 3500. One function of this WBS will be to provide recommendations for future standardization of any APSE component for which there exists a sufficient metrics capability and for which the standardization of such a component is deemed beneficial to the overall Ada program. In addition, this WBS will result in recommendations for the development of tools/aids which will provide or enhance metrics capabilities for identified APSE components.

7.4 4000 APSE Evaluation Capability

- 4100 APSE Evaluation Criteria Analysis

  * This WBS element provides for the review and analysis of existing programming environment evaluation criteria to determine applicability to the E&V Task. This WBS includes review of the Formal Qualification Tests for the existing DoD APSEs. This WBS element also includes review of ongoing standards development activities as a source for criteria development.

- 4200 APSE Formal Qualification Test Monitoring

  * This WBS element provides for the monitoring of the FQT procedures performed on each of the DoD APSEs. Such monitoring will be performed on a non-interference basis, with possible extension to a supporting FQT function.

- 4300 APSE Evaluation Criteria Development

  * This WBS element provides for the development of evaluation criteria which will be applied to existing DoD APSEs. The evaluation criteria developed will be based upon the results of WBS elements 4100 and 4200 and will be included within the E&V Reference Manual developed under WBS 3000.

- 4400 DoD APSE Analysis

  * This WBS element provides for the application of the evaluation criteria developed in WBS element 4300 to existing DoD APSEs. It also provides for an analysis of the features of tools available on each of the DoD APSEs to determine areas of commonality and discrepancy. This analysis will

be performed in concert with an analysis of the NBS Taxonomy effort.

7.5 5000 APSE Validation Capability

- 5100 APSE Validation Analysis

  * This WBS element provides for the review and analysis of existing APSE validation studies to determine applicability to the E&V task. This WBS includes review of validation test suites, such as the ACVC and KAPSE FQT tests.

- 5200 APSE Validation Procedures Analysis

  * This WBS element provides for the review and analysis of existing validation procedures to determine applicability to the E&V Task. This WBS includes review of ACVC procedures, as well as procedures implemented by such organizations as ANSI and ISO.

- 5300 APSE Validation Procedures Development

  * This WBS element provides for the development of validation procedures to be implemented by organizations to which the validation execution responsibility will be transferred.

- 5400 APSE Validation Development

  * This WBS element provides for the development of validation procedures which will be applied to existing DoD APSEs.

- 5500 APSE Validation Application

  * This WBS element provides for the application of the validation procedures, developed in WBS 5400,

to existing DoD APSEs. This WBS also provides for the analysis of results obtained from the application of the validation procedures.

7.6 6000 APSE E&V Tools/Aids

- 6100 APSE E&V Tools/Aids Objectives and Requirements

  * This WBS element provides for the identification of objectives, criteria and requirements to be used for the selection of E&V tools/aids to be acquired/developed as part of the E&V Task. These tools/aids will be used for initial evaluation and/or validation of existing DoD APSEs.

- 6200 APSE E&V Tools/Aids Development Plans

  * This WBS element provides for the analysis necessary to recommend that specific E&V tools/aids be developed. It further provides for making the recommendation, and developing plans for the development and acquisition of these tools/aids.

- 6300 APSE E&V Tools/Aids Development

  * This WBS element provides for the development and acquisition of the recommended APSE E&V tools/aids which will be used for initial evaluation and/or validation of existing DoD APSEs. This WBS includes development of the CAIS Validation Capability (CVC).

- 6400 APSE E&V Tools/Aids Development Review

  * This WBS element provides for the monitoring of the APSE E&V tools/aids development and participation in the APSE E&V tools/aids development review process. It further provides for the reporting of the results of monitoring and reviews.

- 6500 APSE E&V Tools/Aids Application and Analysis

  * This WBS element provides for the overseeing of the application of the E&V tools/aids. It further provides for the development of guidelines for the application of the tools/aids and the analyses of the results produced by their application.

- 6600 APSE E&V Tools/Aids Maintenance

  * This WBS element provides for the maintenance of the APSE E&V Tools/Aids after they are developed.

- 6700 APSE E&V Tools/Aids Modification

  * This WBS element provides for the modification of the APSE E&V Tools/Aids which may be required to correct inadequacies in the first development or to respond to changing requirements.

- 6800 Guidebook for APSE E&V Technology Application

  * This WBS provides for the development of a Guidebook for the application of the E&V technology developed in the E&V Task. The E&V Guidebook will correspond to use of the E&V Reference Manual developed under WBS 3000. This Guidebook will be intended for public use in application to any existing support environment.

7.7 7000 APSE E&V Support

- 7100 APSE E&V Publications

  * This WBS element provides for the publication and distribution of APSE E&V requirements, policy, strategy and other applicable documents.

- 7200 APSE E&V Configuration Management

  * This WBS element provides for the Configuration Mangement of all APSE E&V documents generated and all tools/aids developed in the APSE E&V program.

- 7300 APSE E&V Data Management

  * This WBS element provides for the maintenance, storage and updating of all documentation and data in the APSE E&V program. It further provides for the distribution of all data in the APSE E&V program.

- 7400 APSE E&V Meeting Support

  * This WBS element provides for the technical support required in planning, preparing, conducting and reporting on formal APSE E&V meetings. These meetings are held for the purpose of establishing E&V requirements and an E&V capability.

8. E&V SCHEDULES/MILESTONES

Figure 8-1 illustrates the schedules and milestones associated with the E&V deliverables defined in Section 6 of this document.

Figure 8-2 illustrates the schedules associated with E&V related meetings.

Figure 8-3 illustrates the schedules associated with E&V contractual efforts.

		CY84				CY85				CY86				CY87				CY88				CY89		
	FY84				FY85				FY86				FY87				FY88				FY89			
E&V MILESTONE	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
E&V PLAN	V1 ▲				V2 △				V3 △				V4 △				V5 △				V6 △			
E&V PUBLIC REPORT						V1 △				V2 △				V3 △				V4 △				V5 △		
E&V PROJECT REFERENCE LIST		△																			△			
E&V TECHNICAL COORDINATION STRATEGY				V1 △				V2 △				V3 △				V4 △								
E&V PUBLIC COORDINATION STRATEGY			V1 △				V2 △				V3 △				V4 △									
E&V REQUIREMENTS			V1 △			V2 △			V3 △				V4 △				V5 △							
DoD APSE ANALYSIS				V1 △				V2 △				V3 △				V4 △								
APSE VALIDATION PROCEDURES				V1 △				V2 △				V3 △				V4 △								
E&V CONFIGURATION MANAGEMENT PLAN					△																			
E&V CLASSIFICATION SCHEMA	D ▲				V1 △				V2 △															
E&V REFERENCE MANUAL						V1 △				V2 △				V3 △				V4 △			V5 △			
E&V GUIDEBOOK						V1 △				V2 △				V3 △				V4 △			V5 △			
E&V TOOLS/AIDS REQUIREMENTS					D △		V1 △				V2 △													
E&V TOOLS/AIDS									△											△				
CAIS VALIDATION CAPABILITY									V1 △			V2 △				V3 △				V4 △				

Figure A-8-1. E&V DELIVERABLES

	CY84				CY85				CY86				CY87				CY88				CY89			
	FY84				FY85				FY86				FY87				FY88				FY89			
E&V MILESTONE	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
E&V TEAM MEETINGS	△	△	△	△	△	△	△	△	△	△	△	△	△	△	△	△	△	△	△	△	△	△		
Ada TRI-SERVICE REVIEWS	▲	△	△	△	△	△	△	△	△	△	△	△	△	△	△	△	△	△	△	△	△	△	△	
E&V WORKSHOP			△				△				△				△				△					

Figure A-8-2. E&V MEETINGS

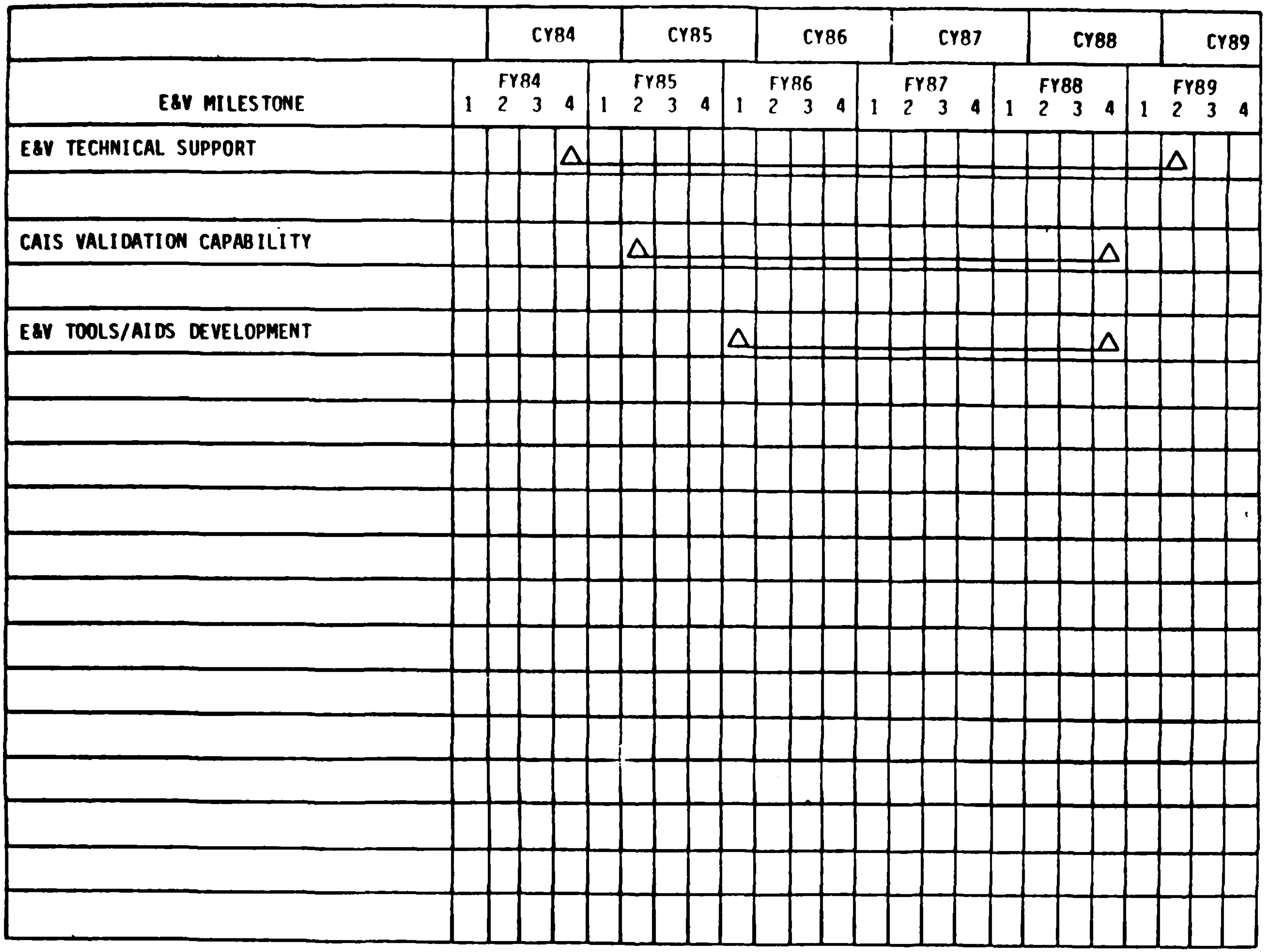


Figure A-8-3. E&V CONTRACTUAL EFFORTS

9. E&V REFERENCES

- [1] Department of Defense, Requirements for High Order Computer Programming Languages -- "STEELMAN", Defense Advanced Research Projects Agency, June 1978.

- [2] Buxton, J.N., Requirements for Ada Programming Support Environments -- "STONEMAN", U.S. Department of Defense, February 1980.

- [3] Houghton, R., "A Taxonomy of Tool Features for the Ada Programming Support Environment (APSE)", U.S. Department of Commerce, National Bureau of Standards, December 1982, Issued February 1983.

- [4] Castor, V., "Criteria for the Evaluation of ROLM Corporation's Ada Work Center", Air Force Wright Aeronautical Laboratories, Wright-Patterson AFB, January, 1983.

- [5] Kafura, D., Lee, J.A.N., Lindquist, T., Probert, T., "Validation in Ada Programming Support Environments", Virginia Polytechnic Institute and State University, December, 1982.

- [6] OUSD (R&E) Memorandum for the Commander, U.S. Army Materiel Development and Readiness Command (DRCDE-SB), Chief of Naval Material (MAT 08Y), Commander Air Force Systems Command (ALR); "Subject: Proposed Evaluation and Validation Tasking", 15 Jun 83.

- [7] HQ Air Force Systems Command (ALR) Letter to AJPO, "Subject: Proposed Evaluation and Validation Tasking (Your Ltr, 15 Jun 83)", 7 Oct 83.

APPENDIX B

REQUIREMENTS FOR EVALUATION AND VALIDATION

OF

ADA PROGRAMMING SUPPORT ENVIRONMENTS

Version 1.0
17 October, 1984

Prepared by

Evaluation and Validation Team
Requirements Working Group

for the
Ada Joint Program Office

Table of Contents

1.0 INTRODUCTION

1.1 Purpose Of The Evaluation And Validation Task

The purpose of the Evaluation and Validation (E&V) Task is to develop the technology which will be used as the basis for the evaluation and validation of Ada Programming Support Environments (APSEs). Validation is the process of determining conformance of an APSE or APSE component to existing standards. For example, Ada compilers are currently required to undergo validation by the Ada Validation Organization (AVO) to insure conformance to the Ada language standard (MIL-STD-1815A). In the future, validation may encompass additional standards such as the Common APSE Interface Set (CAIS) developed by the KAPSE (Kernel APSE) Interface Team/Industry Academic (KIT/KITIA). Evaluation is the process of assessing characteristics or attributes of an APSE or APSE component for which there are no standards. Examples of such attributes include usability, efficiency, and maintainability. In the absence of standards, such attributes are free to vary across different APSE implmentations. Consequently, these attributes are of interest to users in selecting from among different APSE's because they contribute to differences in suitability for different applications or methodologies as well as to differences in overall quality.

It is anticipated that the primary benefits of E&V will be to encourage the development of quality APSE's, to promote interoperability and transportability, and to provide users and developers with a uniform and comprehensive means for assessing and selecting APSE's suitable for their specific applications and methodologies.

1.2 Document Purpose And Scope

The purpose of this document is to set forth requirements on the E&V effort. This document is intended for use by the APSE E&V Team and by the support contractor(s) in developing technology for the evaluation and validation of APSEs.

This document contains three categories of requirements. One category, contained in Section 2, consists of requirements on the E&V Team. These represent requirements against which the organization and activities of the E&V Team can be mapped. They take the form "The E&V Team will..." (e.g., "The E&V Team will develop a validation capability to determine conformance of an APSE to all applicable standards"). A second category, also contained in Section 2, consists of requirements on the E&V methods and procedures. These take the form "The E&V technology shall be..." (e.g., "The E&V technology shall be objective"). The third category, contained in Section 4, consists of requirements on what is to be evaluated and takes the form "The 'X' attribute of the 'Y' component shall be evaluated" (e.g., "The usability of the compiler shall be evaluated").

This document does not contain requirements pertaining to the implementation of E&V technology since these are viewed as falling within the realm of DoD policy and outside the scope of the E&V Team. This document also does not contain requirements on APSE tools, only on the evaluation and validation of those tools.

1.3 Goals

The near term goals are to provide a preliminary or initial set of APSE E&V requirements and a minimal set of E&V tools that can be used to assess APSE components. In addition, a feedback mechanism by which both comments on the tools and requirements and results of applying the tools can be submitted and dissiminated will be developed. It is anticipated that, in the near term, the E&V requirements will be used on existing APSEs and APSE components rather than causing redesign of those components.

The primary long term goal is to establish an interactive database of the results of the performance of E&V on all available APSEs and APSE components. It is expected that this database could be used by both the potential users and the designers of APSE tools. In addition, anyone performing E&V could immediately make the results available to the entire community using the database. It is anticipated that the existence of the E&V database, along with the E&V technology, would have a long term positive effect on the quality of the available Ada support software.

2.0 GENERAL REQUIREMENTS AND CRITERIA FOR THE EVALUATION METHODOLOGY

This document addresses requirements on the E&V Team and on the development of E&V technology. It does not address requirements on the application of E&V technology.

2.1 Requirements On The Evaluation And Validation Team

2.1.1 The E&V Team will develop a validation capability to determine conformance to all applicable standards. This includes the development of tools and aids (e.g., test sets, test scenarios, data reduction capabilities) and other designated means of automated support.

2.1.2 The E&V Team will develop an evaluation capability to assess attributes of APSE components for which no standards exist. This includes the development of tools, aids and other designated means

of automated support.

2.1.3 The E&V Team will generate specific requirements concerning the components and attributes to be evaluated or validated. These requirements will take the following form: "The abc attribute of the xyz component shall be evaluated". Section 4.0 of this document contains illustrative sets of requirements for three major APSE components (compilers, configuration managers, and command language interpreters).

2.1.4 The E&V Team will develop an APSE E&V Classification Schema to guide the generation of specific requirements. For the near term, this schema will take the form of a two-dimensional matrix, with APSE components along one dimension and a list of attributes along the other dimension. Version 1.0 of this matrix appears in Section 4.0

2.1.5 The E&V Team will establish mechanisms for disseminating E&V technology to the public.

2.1.6 The E&V Team will solicit industry and academic participation in the development of E&V technology.

2.1.7 The E&V Team will promote community use and acceptance of E&V technology.

2.1.8 The E&V Team will provide a focal point with respect to APSE E&V.

2.1.9 The E&V Team will maintain expertise on DoD-sponsored APSEs and commercially-sponsored APSE's available within the DoD.

2.1.10 The E&V Team will make recommendations to the DoD on policy decisions affecting the application of E&V technology.

2.1.11 The E&V Team will establish a capability for the independent evaluation of E&V technology to determine and improve the validity and value of the technology which is developed.

2.1.12 The E&V Team will establish E&V product quality guidelines.

2.2 Requirements On Evaluation And Validation Methodology

In outlining requirements on the E&V technology, the following convention is adopted to distinguish between "requirements" and "criteria". Requirements (shalls) distinguish themselves in that fulfillment of the requirement can be clearly observed, while this may not be true for criteria (shoulds).

2.2.1 APSE E&V requirements and the corresponding technology shall be applicable to current APSEs (in order to yield useful results for the short-term), and shall evolve with future APSEs.

2.2.2 E&V shall address individual APSE components and APSEs as a whole.

2.2.3 E&V technology shall be objective. This means that the technology should not be biased toward specific APSE design features or concepts.

2.2.4 E&V technology shall be developed and specified in such a way so as to be repeatable. This means that, within the bounds of statistical sampling error, the same results shall be obtained by two or more evaluations (or validations) conducted independently.

2.2.5 E&V technology shall be comprehensive in that it will consider all important aspects of the component being evaluated or validated.

2.2.6 E&V technology should be relevant, meaning that it should focus on attributes which are important in the development and support of large-scale, embedded systems.

2.2.7 E&V technology shall be tailorable to meet the needs and priorities of specific application areas and organizations.

3.0 APPROACH

3.1 Addressing Requirements On The Evaluation And Validation Team

Although the entire E&V Team is responsible, the primary means of addressing the requirements outlined in Section 2.1 is through the five E&V Team Working Groups.

Requirements 2.1.1 and 2.1.2 are general requirements which serve as the overall charter for the E&V Team. The Common APSE Interface Set Working Group (CAISWG) is currently focusing on CAIS validation while the Requirements Working Group (REQWG) is focusing on evaluation. Requirements 2.1.3 and 2.1.4 are the responsibility of the REQWG and the E&V support contractor respectively.

Requirements 2.1.5, 2.1.6, and 2.1.7 are the responsibility of the Public Coordination Working Group (PUBWG).

Requirement 2.1.8 is the responsibility of the Technical Coordination Working Group (TECWG).

Requirement 2.1.9 is the responsibility of the APSE Analysts Working Group (APSEWG).

Requirements 2.1.10, 2.1.11, and 2.1.12 were addressed by the April, 1984 E&V workshop in Airlie Virginia, and the entire E&V Team is responsible for continued attention to these needs.

3.2 Addressing Requirements On Technology Development

Requirement 2.2.1 will be addressed through the incremental development of E&V technology. An incremental approach will be followed in developing requirements on the E&V Team, requirements on the methods used, and requirements on what is to be evaluated.

For example, the current organizing scheme for generating requirements on what is to be evaluated is contained in Section 4. This scheme takes the form of a component/attribute matrix in which the components represent generic tools existing in current environments. While it will probably continue to be the case that requirements for what is to be evaluated or validated will take the general form of the 'X' attribute of the 'Y' component, the nature and priorities of the attributes are likely to change as will the nature of the components. As an example of a change in the priority of attributes, the ability to interface with other tools will be very important initially since a developing APSE may not include all functionality at an early stage of development; this attribute will become less important over time as more comprehensive toolsets appear. As an example of a change in the nature of the components, with increasing integration of toolsets, components such as compilers will no longer exist as separable entities.

The current list of components (compiler, configuration manager, command language interpreter) provides examples of the approach taken in generating requirements on what is to be evaluated. These components and their sub-components were selected on the basis of expertise within the Team to make an initial attempt at decomposing each component into sub-components and generating questions (contained in the Appendix) for these based on the list of attributes. Thus, the components chosen do not in any way represent a minimally sufficient APSE.

The current organizational scheme has major deficiencies which limit its value for the short term and, even more so, for the long term. There are, for example, alternative ways of decomposing a tool. Not all compilers, command language interpeters, and configuration managers are organized and, hence, decomposable in the ways that are implied by Section 4.0. In addition, identifying the components as generic tools is of limited value, even for the short term. Across different environments, one will not find a common mapping of functions to tools (i.e., tools by the same name do not carry out exactly the same set of functions and the same function is not necessarily contained within a common generic tool).

The longer-term needs for E&V involve the evolution of this document and the development of capabilities that focus on the higher-level units provided by increasing levels of integration. The classification scheme which serves to drive the generation of requirements for E&V will focus more on these higher-level units. The exact form of this classification scheme is currently under discussion. It is likely that multiple classification schemes will result, with increasing emphasis on user perspectives as well as on APSE functionality rather than APSE tools per se. Additional areas of focus for the intermediate and longer term include the following:

1. evaluation of protocols used by functional components

2. development of the CAIS Validation Capability (CVC)

3. definition of semantics and notation for an APSE

4. recommendations on "CAIS-conformance"

5. evaluation of additional functionality, such as simulation/support for Ada-based program description and requirements statement languages

6. development of new procedures and metrics for evaluation

7. refinement of the requirements developed for the short-term

8. increased emphasis on host/target relationship

9. use of E&V early in the APSE life cycle

10. incorporation of new standards/functionality

11. E&V technology inserted into applications level programs

12. increased emphasis on evaluating the system rather than the individual components (Requirement 2.2.2)

13. development of the capability to E&V project-specific, application-specific, methodology-specific APSEs (Requirement 2.2.7)

4.0 REQUIRED APSE EVALUATIONS AND VALIDATIONS

4.1 Introduction

The specific components of an APSE that are candidates for evaluation are listed in this section. As used in this section, components may be tools, features of tools, sets of tools, user-viewable functions performed by the APSE, facilities provided by the APSE and used by some other component, or any software that provides one of the four interface classes defined in the E&V Plan [1]. This section divides the requirements for evaluating and validating APSE components into two categories, component requirements and macroscopic requirements. Component evaluations and validations detail the technology needed to address individual APSE components exclusive of the remainder of the environment. Requirements in this category include those such as compiler and editor evaluations. Component-wise requirements are detailed in Section 4.3, and these are based on the attribute definitions of Section 4.2. Macroscopic evaluations and validations detail the

technology needed to address the interactions among APSE components and technology that addresses all APSE components independent of their function. Included in these requirements are evaluations of the human-to-APSE interface and intertool data interfaces.

4.2 Attribute Definitions

Component-wise requirements are based on the set of attributes that the component is to possess. To provide a consistent meaning, the following attribute definitions and interpretations have been adopted for E&V use.

1. Availability - The probability that a component will be functionally ready or operable at some specified point in time. [14]

2. Capacity - The upper or lower limit of a component's functions or features.

3. Completeness - The extent to which a tool provides the complete set of operations necessary to perform a job. For example, the capability to undo an operation in case of error.

4. Configuration Management - All activities related to controlling the contents of a component; including monitoring the status, preserving the integrity of released and developing versions, and controlling the effects of changes throughout the component. [14]

5. Correctness - Agreement between a component's total response and the stated response in the functional specification (functional correctness), and/or between the component as coded and the programming specification (algorithmic correctness). [14]

6. Costs - The costs of a completed component or the costs features of a component. Costs of a component may vary depending on delivery with source code or object code only (for example). Other cost considerations are installation, user assistance, and maintenance support.

7. Documentation - The technical data, including on-line, computer listings, and printouts, which serve the purposes of: (1) elaborating the design or details of a component, (2) explaining the capabilities of a component, and (3) providing operating instructions for using the component to obtain desired results. [14]

8. Efficiency - The extent to which a component fulfills its purpose using a minimum of computing resources. This implies that choices of source code constructions are made in order to produce the minimum number of words of object code, or that where alternate algorithms are

available, those taking the least time are chosen; or that information-packing density in core is high, etc. Of course, many of the ways of coding efficiently are not necessarily efficient in the sense of being cost-effective, since portability, maintainability, etc., may be degraded as a result. [14]

9. Extendability (Extensibility) - The extent to which a component allows new capabilities to be added and existing capabilities to be easily tailored to user needs. [14]

10. Granularity - The degree to which a component has separate limited functions that are composable, user selectable, and communicate through a common data base.

11. Hardware - Design and implementation features of a component which take advantage of host or target dependent hardware techniques and performance.

12. Integrity - Extent to which access to a component or associated data by unauthorized persons can be controlled.

13. Interfaces - The common boundary between software components, between hardware devices, or between hardware and software. When applied to software components, that set of assumptions made concerning the component by the remaining components and the system in which it appears. Software components have control, data, and services interfaces. Included in this attribute is conformance to any existing pertinent interface standards. [14]

14. Interoperability - The ability of APSEs to exchange data base objects and their relationships in forms usable by components and user programs without conversion. Interoperability is measured in the degree to which this exchange can be accomplished without conversion.

15. Intraoperability - The ability of APSE components to exchange data base objects and their relationships in forms usable without conversion.

16. Maintainability - The extent to which a component facilitates updating to satisfy new requirements or to correct deficiencies. This implies that the it is understandable, testable, and modifiable. [14]

17. Maturity - The extent to which a component has been used in the development of deliverable software by typical users and to which the feedback from that use has been reflected in modifications to it.

18. Transportability - The ability of a component to be installed on a different APSE without change in functionality. Transportability is measured in the degree to which this installation can be accomplished without reprogramming.

19. Power - The extent to which a component has capabilities such as default options and wild card operations.

20. Proprietary - Restrictions on the release, distribution, or use of a component. This includes so called "data rights" restrictions.

21. Rehostability - The ability of an APSE component to be installed on a different host with needed reprogramming localized to the KAPSE or machine dependencies.

22. Reliability - The extent to which a component can be expected to perform its intended functions in a satisfactory manner. [14]

23. Resources Required - The amount and types of hardware or software facilities needed for the operation of a component. This includes primary and secondary storage and any other required resources.

24. Retargetability - The ability of an APSE component to accomplish its function with respect to another target. The component may require modification.

25. Reusability - Extent to which a component can be used in other applications; related to the packaging and scope of the functions that components perform. [14]

26. Robustness - (1) Protection of a component from itself, user, and system errors. The ability to recover and provide meaningful diagnostics in the event of unforeseen situations.

27. Software Production Vehicle - The methodology(ies), language(s), and technique(s) used to produce the software related to a component.

28. Test Availability - The availability of tests which verify the correctness or effectiveness of a component function or feature. These tests may also verify proper response for an incorrect input or technique.

29. Testability - The extent to which a component facilitates the establishment of verification criteria and supports evaluation of its performance. This implies that requirements are matched to specific modules, or diagnostic capabilities are provided, etc. [14]

30. Usability - User effort required to learn, operate, prepare input, and interpret output of a component.

NOTE: The definitions which reference other literature are not taken verbatim from the references. The definition may have been changed to conform to E&V objectives.

4.3 Required Component Evaluations

The component-wise requirements of this section are derived from a matrix of APSE components and attributes. Each Subsection of Section 4.3 addresses a different APSE component. Currently, the set of components addressed are taken from the MAPSE (Minimal Ada Programming Support Environment) toolset outlined in STONEMAN [10]. These include:

1. Command Language Interpreter capable of invoking all APSE tools.
2. Compiler capable of translating source Ada programs into target code for the host and at least one target.
3. Configuration Manager capable of assisting in long term configuration control of projects.
4. Control Flow Static Analyzer capable of producing a chart of the program control topology.
5. Dynamic Analysis Tool, on systems with an interactive capability, capable of providing the following functions: snap shot, break, trace, interface simulator, statement execution monitor, and timing analysis.
6. File Administrator capable of transfering and comparing files.
7. Linkers, for both the host and target machines, capable of partial linking of program units and creating executable programs from program units.
8. Loaders capable of off-line and/or down-line loading.
9. Prettyprinter capable of printing database objects in legible formats which depend on the object categorization.
10. Set-Use Static Analyser capable of providing a cross-reference map indicating where each data item is changed in value and where it is merely referenced.
11. Text Editor capable of editing general text, including specifications, design and other documents, and source programs.

Each Subsection presents a hierarchical breakdown of the component, followed by an explanation of the hierarchy elements. Requirements for addressing certain attributes (from Section 4.2) of hierarchy elements are listed. The format of component-wise requirements is abbreviated as follows:

Compiler/Efficiency

to refer to the compiler hierarchy element-Compiler and the attribute-Efficiency. This abbreviation represents the requirement:

The efficiency attribute of the compiler component shall be evaluated.

Each requirement, listed in the abbreviated *form above,* is augmented by one or more questions which address the requirement. While these questions may later become part of the Evaluation and Validation Team Reference Manual, they are currently included in Appendices to demonstrate the requirement.

NOTE: IN THE CURRENT VERSION, EACH REQUIRED EVALUATION IS LISTED SEPARATELY IN TEXT IN THE FORMAT SHOWN ABOVE. IN A LATER VERSION OF THE DOCUMENT, THE TEXT WILL BE REPLACED BY A TABULATED ARRANGEMENT ALLOWING A MORE CONCISE PRESENTATION OF THE REQUIREMENTS.

4.3.1 Command Language Interpreter -

The command language interpreter is probably the only component of an Ada Programming Support Environment to which all of the users of the APSE are exposed. While other APSE components, such as the compiler, will be more extensively used, every user of an APSE, including those not directly involved in software development, such as managers, must at least use the command language interpreter once to invoke the APSE function which they use. In addition, it is only through the command language interpreter that the user of an APSE can cause the different components or tools within the APSE to interact with each other. Thus, the the command language interpreter, more than most other functions, goes a long way towards establishing the character of an APSE.

In spite of the importance of the command language interpreter, STONEMAN has only the following to say on the subject:

"*4.D.4 COMMAND LANGUAGES: The requirements of* 4.C.3 (individual tool invocation) and 4.C.4 (virtual interface from a variety of physical terminal devices) may well be implemented by a command language (or job control language).

"Regardless of the choice of command language, the environment must provide a primitive operation which enables the initiation of a program to be carried out. More precisely, this operation permits a data structure (such as a compiler output) to be executed as a program on the host.

"Given this primitive, one possible approach to the implementation of a command language is to use a basic Ada-like language whose facilities, offered by a simple interpreter tool, provide little more than the ability to perform simple editing of command lines and to initiate programs.

"The requirement in 4.C.6 (APSE tools may invoke other tools) indicates that the primitive initiation facility used by the command language will be made available as a library procedure to Ada programs. This will enable the user to construct job control sequences as Ada program texts which initiate other programs. This use may well be subject to some restrictions; for example, to prevent recursive initiation in unsuitable cases.

"A more general approach is to regard the user interaction as being expressed entirely within Ada program segments which are executed or interpreted as necessary in the context of relevant points in the APSE database, thus providing a total Ada environment similar, for example, to an Interlisp environment."

Thus, it is clear that, beyond requiring that both users and programs be able to invoke programs, STONEMAN does not even explicitly require a command language but merely states that there "may well" be one. The real effect of all this is that STONEMAN actually allows maximal latitude in the area of the command language. This is summarized by the final statement in STONEMAN on command languages:

"... In view of this range of possibilities, the detailed choice of command language is left as a design decision for the specific APSEs."

Although STONEMAN only specifically requires a relatively crude "command language" with the primitive program invocation function, it is expected that the command languages implemented in actual APSEs will be considerably more sophisticated. The authors of STONEMAN appeared to assume that the requirement that programs be able to invoke other programs would result in an Ada-like command language. While the core of many command languages will undoubtably be Ada-like, it is still possible for a command language to use inputs that are not only not Ada-like but not even tabular in nature; for example, graphical inputs or menu driven capabilities. In addition, it can be expected that some command languages will have high levels of intelligence, including such capabilities as "do what I mean" and "undo."

It is within this framework of minimal expressed requirements with unbounded possibilities that the following functional hierarchy is introduced.

Command Language Interpreter Hierarchy

1. Command language interpreter
2. Command language
3. Syntax
4. Programs
5. Tool/program invoking function
6. Diagnostic generation function
7. Non-tabular inputs
8. Interpreter
9. Hosts
10. Interfaces
11. Aids
12. Performance

Definitions. Following are the Command Language Interpreter components given in the hierarchy.

1. Command language interpreter. The command language of an APSE together with the APSE tool which implements it.

2. Command language. The set of command sequences which can be recognized by the command language interpreter. This can include tabular imputs which may or may not be Ada-like in appearance or non-tabular inputs such as graphical inputs, selections from menus or device dependent responses.

3. Syntax. The format of tabular command sequences including the nature of identifiers, control constructs and other elements.

4. Programs. Complete, self contained command sequences which can be executed from other command sequences or user programs.

5. Tool/program invoking function. The ability to invoke an APSE tool or program from a command sequence. This is the primary command language requirement imposed by STONEMAN.

6. Diagnostic generation function. The ability for a command language sequence to generate diagnostic messages for use by the user of a command language program.

7. Non-tabular inputs. Command language inputs which are not in the form of programming language-like statements, such as graphical inputs or direct device specific hardware responses.

8. Interpreter. The APSE tool or function which actually interprets or executes the command language sequences. It will probably be a software program itself.

9. Hosts. The computer(s) upon which the command language interpreter, and presumably the APSE itself, is implemented.

10. Interfaces. The interfaces between the command language interpreter and the other APSE tools and programs.

11. Aids. Facilities for assisting the user of the command language interpreter; for example, help facilities, diagnostic messages, formatter and prompter.

12. Performance. Characteristic capacities and response times of the command language interpreter.

Command Language Interpreter Requirements

The Availability of the Command Language Interpreter shall be evaluated.

The Capacities of the Command Language Interpreter shall be

evaluated.

The Configuration Management of the Command Language Interpreter shall be evaluated.

The Costs of the Command Language Interpreter shall be evaluated.

The Documentation of the Command Language Interpreter shall be evaluated.

The Extendability of the Command Language Interpreter shall be evaluated.

The Interfaces of the Command Language Interpreter shall be evaluated.

The Interoperability of the Command Language Interpreter shall be evaluated.

The Maintainability of the Command Language Interpreter shall be evaluated.

The Proprietary Rights of the Command Language Interpreter shall be evaluated.

The Test Availability of the Command Language Interpreter shall be evaluated.

The Maturity of the Command Language Interpreter shall be evaluated.

The Extendability of the Command Language shall be evaluated.

The Robustness of the Command Language shall be evaluated.

The Usability of the Command Language shall be evaluated.

The Usability of the Syntax shall be evaluated.

The Extendability of the Syntax shall be evaluated.

The Extendability of Programs shall be evaluated.

The Granularity of Programs shall be evaluated.

The Interfaces of Programs shall be evaluated.

The Interoperability of Programs shall be evaluated.

The Flexibility of Programs shall be evaluated.

The Usability of the Tool/Program Invoking Function shall be evaluated.

The Interoperability of the Tool/Program Invoking Function shall be evaluated.

The Flexibility of Non-tabular Inputs shall be evaluated.

The Interoperability of the Interpreter shall be evaluated.

The Resources Required by the Interpreter shall be evaluated.

The Availability of Hosts shall be evaluated.

The Rehostability of Hosts shall be evaluated.

The Interoperability of Interfaces shall be evaluated.

The Usability of Aids shall be evaluated.

The Efficiency of the Performance shall be evaluated.

4.3.2 Compiler -

It is important to realize the need for evaluating Ada compilers in light of a comprehensive, formal validation process which each compiler must pass before it may be used in official software development projects. The validation of Ada compilers is performed by executing the Ada Compiler Validation Capability (ACVC) suite of tests. These tests measure conformance of the compiler to the Ada language standard. There are a number of requirements and issues which cannot always be measured by a "pass" or "fail" criteria. In many cases, these are more difficult to measure and require careful study and evaluation. These include:

1. Compiler implementation and design techniques to verify an efficient and usable product.
2. Performance, efficiency, optimizations, options and capacities.
3. The existence and quality of user and system documentation.
4. Configuration management and control procedures for documentation, source code, object code, compiler baselines, versions, and tracking of failure and error recovery fixes.
5. Compiler maintenance supportability and procedures.

Compiler Hierarchy

1. Compiler
2. Input
3. Command Language
4. User Assistance
5. Source Statements
6. Translation
7. Analysis
8. Intermediate Forms
9. Optimization
10. Symbol Table

11. Code Generation
12. Debugging
13. Optimization
14. Output
15. Analysis
16. Cross-Reference
17. Listing
18. Object Module
19. Run Time System
20. Memory Management
21. Task Management
22. Distributed Processing
23. Parallel Processing
24. Exception Handling
25. Data Management
26. Mathematical Functions

Definitions. Following are the definitions for compiler components given in the hierarchy.

1. Compiler - Issues which are applicable to the entire compiler system, rather than a component of the compiler.

2. Input - User inputs to the compiler.

3. Command Language - The language with which the user invokes the compiler, including various user options. Mode of invocation, such as interactive or batch are considered.

4. User Assistance - Built-in compiler user assistance such as invocation help, and user debugging facilities.

5. Source Statements - The Ada computer program as input by the user.

6. Translation - The compiler process of transforming the source statements into a form suitable for code generation.

7. Analysis - Compiler analysis during the translation process. This includes error analysis, user aid analysis (e.g., helpful warning messages) and statistical analysis.

8. Intermediate Forms - Translation of the user program into forms which aid the transformation and final code generation phases.

9. Optimization - Compiler techniques which result in better run time efficiency or less memory than would have been possible if the optimizations had not been performed.

10. Symbol Table - An internal compiler data structure which gives attributes, types and other needed compiler information on program identifiers and any other items of interest to the compiler.

11. Code Generation - The process of transforming the final intermediate language form into machine language instruction code sequences for a particular target (or host). Since the primary function of a compiler is to generate object code, it is useful to discuss the requirements of code quality. The quality of the produced machine instructions affects both the memory size and efficiency of the executable program. Most code generators produce acceptable code for language constructs such as arithmetic and assignments. However, many implementation decisions and language features may be designed in a number of ways. A few of these include register usage, techniques of passing parameters, movement of large amounts of data, subroutine calling and returning sequences, stack management, and memory management. In the past, most users simply accepted the produced machine code (as long as there were no compiler bugs). The use of Ada in embedded computer systems in real-time and/or tasking applications will be for a sophisticated and knowledgeable class of users who require and demand efficient machine code. A number of studies have been done comparing the efficiency of compiler generated code to hand coded assembly language. In general, these studies have shown that an experienced assembly language programmer can generate code which is 10-25% more efficient than the compiled code. One needs to be extremely careful in using these comparisons. The study may be biased to one party or another. The benchmarks that were used may not be representative of the general application, or purposely chosen to bias the results. Despite all of the above, a set of carefully chosen benchmarks (identically coded in both Ada and assembler) need to be executed to provide an estimate of compiler code generation efficiency.

12. Debugging - If a compiler design includes user debugging capabilities, the "hooks" for this feature are implemented in the code generation phase. Therefore, this component addresses techniques of debugging implementation.

13. Optimization - This component deals with optimization which is specific to the target (or host). In general, this optimization takes advantage of machine dependent features.

14. Output - Outputs produced by the compiler that are usable by the user or other APSE tools or components.

15. Analysis - Outputs as a result of earlier compiler analysis, such as error messages, statistics, debugging information, and intermediate forms.

16. Cross-Reference - A formatted output containing ordered information. The information includes identifier attributes, types and statement numbers given for where used, set, etc. Other information may be given.

17. Listing - An image of the source program along with compiler statement numbers, error messages, and other information useful to the user.

18. Object Module - (Sometimes called the relocatable module). The machine language instruction representation of the source program. Also included is target dependent information needed by the linker or linkage editor.

19. Run Time System - (Sometimes referred to as the run time environment). Includes those modules that are not considered to be needed for translation, but are needed during the execution. They generally perform operations peculiar to a particular host of target machine. It should be noted that different compilers may perform more or fewer of the functions than given herein. Therefore, the run time system should be evaluated based on the machine and intended application. Since the Ada language system was primarily developed for use in embedded computer systems, there are a number of issues and considerations for the run time system that may not apply to other languages. One of the unique features of Ada is the tasking capability. This implies that ability to perform parallel processing on two or more CPUs. The technology to perform parallel processing is still in its infancy (a great deal of research is in progress). The tasking issues themselves (using just one processor) present a number of challenges. In the past, it was generally accepted that the functions of tasking would be handled by the host or target operating system. Embedded applications developed their own real-time operating system (executive) to handle the functions. Since the high level language did not contain any tasking language constructs, the compiler implementors were simply not concerned with this issue. With Ada, the compiler implementor is faced with generating code to perform these operations. Several questions immediately arise: 1) Does the compiler designer possess sufficient skills to implement this technology, and should he be expected to do it? 2) Considering the many potential target systems, is the technology available to implement those features in a generalized (but efficient) manner, or does each target require a substantially different run time system implementation? 3) Will the end user accept the compiler designer's product, or should the user be responsible for the run time system for these functions? It is doubtful that questions such as these will be resolved easily. It appears at this point that one solution is a close (hopefully friendly) working relationship between the compiler designers and the technical staff of the user.

20. Memory Management - Allocation or freeing of memory to perform functions needed to implement language features requiring memory management. Also included is memory management during the compilation phase.

21. Task Management - Modules which implement the tasking constructs of the Ada language.

22. Distributed Processing - Implementation of the ability for a program or system to execute on more than one processor. Program execution may or may not be in parallel.

23. Parallel Processing - Implementation of the ability to execute a program or systems functions in parallel on more than one processor.

24. Exception Handling - Facilities for dealing with errors or other exceptional situations that arise during program execution (run time) [7]. Note that this does not include compile time errors.

25. Data Management - Modules dealing with data mappings and manipulations peculiar to the host or target. This includes input/output implementations.

26. Mathematical Functions - Functions, subprograms, or packages which perform operations of a scientific, statistical or specialized nature.

Compiler Requirements

The Availability of the Compiler shall be evaluated.

The Capacity of the Compiler shall be evaluated.

The Configuration Management of the Compiler shall be evaluated.

The Cost of the Compiler shall be evaluated.

The Documentation of the Compiler shall be evaluated.

The Efficiency of the Compiler shall be evaluated.

The Extendability of the Compiler shall be evaluated.

The Granularity of the Compiler shall be evaluated.

The Hardware of the Compiler shall be evaluated

The Interfaces of the Compiler shall be evaluated.

The Interoperability of the Compiler shall be evaluated.

The Maintainability of the Compiler shall be evaluated.

The Proprietary Rights of the Compiler shall be evaluated.

The Rehostability of the Compiler shall be evaluated.

The Retargetability of the Compiler shall be evaluated.

The Resources Required by the Compiler shall be evaluated.

The Robustness of the Compiler shall be evaluated.

The Test Availability of the Compiler shall be evaluated.

The Usability of the Compiler shall be evaluated.

The Availability of Input shall be evaluated.

The Capacity of Input shall be evaluated.

The Documentation of Input shall be evaluated.

The Usability of Input shall be evaluated.

The Availability of the Input, Command Language shall be evaluated.

The Documentation of the Input, Command Language shall be evaluated.

The Test Availabiltiy of the Input, Command Language shall be evaluated.

The Availability of Input, User Assistance shall be evaluated.

The Documentation of Input, User Assistance shall be evaluated.

The Capacity of Input, Source Statements shall be evaluated.

The Usability of Input, Source Statements shall be evaluated.

The Availability of Translation shall be evaluated.

The Documentation of Translation shall be evaluated.

The Efficiency of Translation shall be evaluated.

The Extendability of Translation shall be evaluated.

The Interfaces of Translation shall be evaluated.

the Interoperability of Translation shall be evaluated.

The Maintainability of Translation shall be evaluated.

The Proprietary Rights of Translation shall be evaluated.

The Rehostability of Translation shall be evaluated.

The Resources Required of Translation shall be evaluated.

The Robustness of Translation shall be evaluated.

The Test Availability of Translation shall be evaluated.

The Avaliablity of the Translation, Analysis shall be evaluated.

The Documentation of the Translation, Analysis shall be evaluated.

The Efficiency of the Translation, Analysis shall be evaluated.

The Granularity of the Translation, Analysis shall be evaluated.

The Interoperability of the Translation, Analysis shall be evaluated.

The Proprietary Rights of the Translation, Analysis shall be evaluated.

The Test Availabiltiy of the Translation, Analysis shall be evaluated.

The Usability of the Translation, Analysis

The Availability of the Translation, Intermediate Forms shall be evaluated.

The Documentation of the Translation, Intermediate Forms shall be evaluated.

The Extendability of the Translation, Intermediate Forms shall be evaluated.

The Interfaces of the Translation, Intermediate Forms shall be evaluated.

The Interoperabiliyt of the Translation, Intermediate Forms shall be evaluated.

The Proprietray Rights of the Translation, Intermediate Forms shall be evaluated.

The Rehostabiltiy of the Translation, Intermediate Forms shall be evaluated.

The Test Availability of the Translation, Intermediate Forms shall be evaluated.

The Usability of the Translation, Intermediate Forms shall be evaluated.

The Availability of the Translation, Optimization shall be evaluated.

The Documentation of the Translation, Optimization shall be evaluated.

The Extendability of the Translation, Optimization

The Rehostability of the Translation, Optimization shall be evaluated.

The Test Availability of the Translation, Optimization shall be evaluated.

The Usability of the Translation, Optimization shall be evaluated

The Capacity of the Translation, Symbol Table shall be evaluated

The Documentation of the Translation, Symbol Table shall be evaluated.

The Efficiency of the Translation, Symbol Table shall be

evaluated.

The Interoperabiliyt of the Translation, Symbol Table shall be evaluated.

the Resources Required of the Translation, Symbol Table shall be evaluated.

The Test Availability of the Translation, Symbol Table

The Usability of the Translation, Symbol Table shall be evaluated.

The Availability of the Code Generation shall be evaluated.

The Capacity of the Code Generation shall be evaluated.

The Documentation of the Code Generation shall be evaluated.

The Efficiency of the Code Generation shall be evaluated.

The Granularity of the Code Generation shall be evaluated.

The Hardware of the Code Generation shall be evaluated.

The Interfaces of the Code Generation shall be evaluated.

The Interoperability of the Code Generation shall be evaluated.

The Maintainability of the Code Generation shall be evaluated.

The Proprietary Rights of the Code Generation shall be evaluated.

The Rehostability of the Code Generation shall be evaluated.

The Retargetability of the Code Generation shall be evaluated.

The Test Availability of the Code Generation shall be evaluated.

The Usability of the Code Generation shall be evaluated.

The Availability of the Code Generation, Debugging shall be evaluated.

The Documentation of the Code Generation, Debugging shall be evaluated.

The Efficiency of the Code Generation, Debugging shall be evaluated.

The Granularity of the Code Generation, Debugging shall be evaluated.

The Hardware of the Code Generation, Debugging shall be evaluated.

The Interfaces of the Code Generation, Debugging shall be evaluated.

The Interoperability of the Code Generation, Debugging shall be evaluated.

The Proprietary Rights of the Code Generation, Debugging shall be evaluated.

The Rehostabiliy of the Code Generation, Debugging shall be evaluated.

The Retargetability of the Code Generation, Debugging

The Resources Required of the Code Generation, Debugging shall be evaluated.

The Test Availability of the Code Generation, Debugging shall be evaluated.

The Usability of the Code Generation, Debugging shall be evaluated.

The Availability of the Code Generation, Optimization shall be evaluated.

The Documentation of the Code Generation, Optimization shall be evaluated.

The Efficiency of the Code Generation, Optimization shall be evaluated.

The Hardware of the Code Generation, Optimization shall be evaluated.

The Proprietary Rights of the Code Generation, Optimization shall be evaluated.

The Retargetability of the Code Generation, Optimization shall be evaluated.

The Test Availability of the Code Generation, Optimization shall be evaluated.

The Usability of the Code Generation, Optimization shall be evaluated.

The Availability of Output shall be evaluated.

The Documentation of Output shall be evaluated.

The Hardware of Output shall be evaluated.

The Interfaces of Output shall be evaluated.

The Resources Required of Output shall be evaluated.

The Test Availability of Output shall be evaluated.

The Usability of Output shall be evaluated.

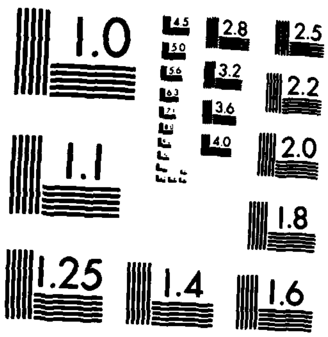
The Availability of the Output, Analysis shall be evaluated.

The Documentation of the Output, Analysis shall be evaluated.

The Extendability of the Output, Analysis shall be evaluated.

AD-A153 609 EVALUATION AND VALIDATION (E&V) TEAM PUBLIC REPORT 2/6
VOLUME 1(U) AIR FORCE WRIGHT AERONAUTICAL LABS
WRIGHT-PATTERSON AFB OH V L CASTOR 30 NOV 84

UNCLASSIFIED AFWAL-TR-85-1016-VOL-1 F/G 14/2 NL



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

The Granularity of the Output, Analysis shall be evaluated.

the Interfaces of the Output, Analysis shall be evaluated.

The Interoperability of the Output, Analysis shall be evaluated.

The Rehostability of the Output, Analysis shall be evaluated.

The Test Availability of the Output, Analysis shall be evaluated.

The Usability of the Output, Analysis shall be evaluated.

The Availability of the Output, Cross-Reference shall be evaluated.

The Documentation of the Output, Cross-Reference shall be evaluated.

The Test Availability of the Output, Cross-Reference shall be evaluated.

The Usability of the Output, Cross-Reference shall be evaluated.

The Documentation of the Output, Listing shall be evaluated.

The Test Availability of the Output, Listing shall be evaluated.

The Usability of the Output, Listing shall be evaluated.

The Documentation of the Output, Object Module shall be evaluated.

The Interfaces of the Output, Object Module shall be evaluated.

The Rehostability of the Output, Object Module shall be evaluated.

The Retargetability of the Output, Object Module shall be evaluated.

The Test Availability of the Output, Object Module shall be evaluated.

The Usability of the Output, Object Module shall be evaluated.

The Availability of the RTS(Run Time System) shall be evaluated.

The Documentation of the RTS shall be evaluated.

The Efficiency of the RTS shall be evaluated.

The Extendability of the RTS shall be evaluated.

The Granularity of the RTS shall be evaluated.

The Hardware of the RTS shall be evaluated.

The Interfaces of the RTS shall be evaluated.

The Maintainability of the RTS shall be evaluated.

The Proprietary Rights of the RTS shall be evaluated.

The Rehostability of the RTS shall be evaluated.

The Retargetability of the RTS shall be evaluated.

The Resources Required of the RTS shall be evaluated.

The Robustness of the RTS shall be evaluated.

The Test Availability of the RTS shall be evaluated.

The Usability of the RTS shall be evaluated.

The Availability of the RTS, Memory Management shall be evaluated.

The Capacity of the RTS, Memory Management shall be evaluated.

The Documentatio of the RTS, Memory Management shall be evaluated.

The Efficiency of the RTS, Memory Management shall be evaluated.

The Hardware of the RTS, Memory Management shall be evaluated.

The Interfaces of the RTS, Memory Management shall be evaluated.

The Test Availability of the RTS, Memory Management shall be evaluated.

The Usability of the RTS, Memory Management shall be evaluated.

The Availability of the RTS, Task Management shall be evaluated.

The Capacity of the RTS, Task Management shall be evaluated.

The Documentation of the RTS, Task Management shall be evaluated

The Efficiency of the RTS, Task Management shall be evaluated.

The Interfaces of the RTS, Task Management shall be evaluated.

The Robustness of the RTS, Task Management shall be evaluated.

The Test Availability of the RTS, Task Management shall be evaluated.

The Usability of the RTS, Task Management shall be evaluated.

The Availabilty of the RTS, Task Management, Distributed Processing shall be evaluated.

The Capacity of the RTS, Task Management, Distributed Processing shall be evaluated.

The Documentation of the RTS, Task Management, Distributed Processing shall be evaluated.

The Efficiency of the RTS, Task Management, Distributed Processing shall be evaluated.

The Hardware of the RTS, Task Management, Distributed Processing shall be evaluated.

The Resources Required of the RTS, Task Management, Distributed Processing shall be evaluated.

The Robustness of the RTS, Task Management, Distributed Processing shall be evaluated.

The Test Availability of the RTS, Task Management, Distributed Processing shall be evaluated.

The Availability of the RTS, Task Management, Parallel Processing shall be evaluated.

The Capacity of the RTS, Task Management, Parallel Processing shall be evaluated.

The Documentation of the RTS, Task Management, Parallel Processing shall be evaluated.

The Rehostability RTS, Task Management, Parallel Processing shall be evaluated.

The Retargetability of the RTS, Task Management, Parallel Processing shall be evaluated.

The Test Availability of the RTS, Task Management, Parallel Processing shall be evaluated.

The Availability of the RTS, Exception Handling shall be evaluated.

The Documentation of the RTS, Exception Handling

The Efficiency of the RTS, Exception Handling shall be evaluated.

The Hardware of the RTS, Exception Handling shall be evaluated.

The Interfaces of the RTS, Exception Handling shall be evaluated.

The Test Availability of the RTS, Exception Handling shall be evaluated.

The Usability of the RTS, Exception Handling shall be evaluated.

The Availability of the RTS, Data Management shall be evaluated.

The Capacity of the RTS, Data Management shall be evaluated.

The Documentation of the RTS, Data Management shall be evaluated.

The Efficiency of the RTS, Data Management shall be evaluated.

The Hardware of the RTS, Data Management shall be evaluated.

The Interfaces of the RTS, Data Management shall be evaluated.

The Rehostability of the RTS, Data Management shall be evaluated.

The Retargetability of the RTS, Data Management shall be evaluated.

The Robustness of the RTS, Data Management shall be evaluated.

The Test Availability of the RTS, Data Management shall be evaluated.

The Usability of the RTS, Data Management shall be evaluated.

The Availability of the RTS, Mathematical Functions shall be evaluated.

The Documentatio of the RTS, Mathematical Functions shall be evaluated.

The Efficiency of the RTS, Mathematical Functions shall be evaluated.

The Extendability of the RTS, Mathematical Functions shall be evaluated.

The Interfaces of the RTS, Mathematical Functions shall be evaluated.

The Proprietary Rights of the RTS, Mathematical Functions shall be evaluated.

The Rehostability of the RTS, Mathematical Functions shall be evaluated.

The Resources Required of the RTS, Mathematical Functions shall be evaluated.

The Test Availability of the RTS, Mathematical Functions shall be evaluated.

The Usability of the RTS, Mathematical Functions shall be evaluated.

4.3.3 Configuration Management -

Software configuration management (CM) is an information processing function involving many people in a software development organization and applying to many different objects. The manner in which it is done is dependent on at least the software development organization, the development methodologies employed, and the characteristics of the software under management. The appropriate tools to support configuration management are therefore somewhat dependent on the particular circumstance in which they will be used.

The Stoneman requirements for configuration management are limited to the very minimal set summarized as follows:

> The Stoneman requires that the MAPSE must contain a configuration management tool that will "assist in the long term configuration control of projects. As minimal

functions this tool will enable interrogation of history attributes and will offer managerial control over the persistence of objects in the database." [10, 6.A.12] The APSE configuration management tool must be capable of determining the origin and purpose of each configuration and of controlling the further development and maintenance of the configuration. [10,2.B.5(8)] In order to provide these capabilities, the KAPSE must maintain a history attribute recording the manner in which the configured item was produced and all information which was relevant in its production. [10,5.A.5] Also, the configuration management tool must communicate via the central database provided by the KAPSE. This database will store all relevant information concerning a project throughout its life cycle. The database will be structured so that relationships between objects in the database can be maintained for configuration control. [10,2.B.4] The detailed requirements of a configuration management tool are left open to a large degree to be tailored according to the organization using it [10,4.B.4].

In the absence of detailed requirements for a MAPSE configuration management tool and of a working consensus as strong as there is for some other tools, the functional decomposition herein must be regarded as a minimal set of primitive functions, tentatively proposed. Further work in the definition of concepts and terms for software configuration management and subsequently in the proposing of model requirements will be essential to the development of useful evaluation methods.

The material in this section draws heavily from references [8, 9 and 10.]

Configuration Management Component Hierarchy

1 Configuration Management
2 Identification
3 Attribute management
4 Version management
5 Variation management
6 Relationship management
7 Configuration Control
8 Workspace partitioning
9 Access control
10 Baseline management
11 Protection
12 Status Accounting and Reporting
13 History reporting
14 Configuration reporting

<u>Definitions</u>. Following are the definitions for the Configuration Management components given in the hierarchy.

Configuration Item, Configuration, and Baseline. A configuration item is a data base object that is under configuration management. The data base object may be simple (e.g., source code for a module) or complex (e.g.,

the definition of a software configuration or a baseline.) A configuration is an aggregation of components and of the relationships among them. [9] A baseline is the content of a configuration at a designated and fixed time during the software life cycle. [9]

1. Configuration Management. The software configuration management function consists of identifying and documenting the characteristics of a configuration item, controlling changes to the characteristics, and recording and reporting the changes to and status of configuration items. [9]

2. Identification. The configuration identification function establishes a definition and identification of the functional and physical characteristics of any configured item. It identifies the relationship between configured items and insures a consistent generation of identification labels for subsequent versions of modified configured items.

3. Attribute Management. Attribute management is the function of creating, changing (both automatically and at the command of the user), and providing access to attributes of configuration management data base objects.

4. Version Management. Version management is comprised of the generating of a revision group composed of successive versions of a database object, generating a unique name for each version, and accessing members of the group directly and implicitly.

5. Variation Management. Variation management is comprised of the generation of variations of a database object (e.g., object code modules for different targets), each with a unique name, and the accessing of a variation directly and implicitly.

6. Relationship Management. Relationship management consists of creating, changing (automatically or by user command), and providing access to relationships between configuration management data base objects.

7. Configuration Control. Configuration control manages access to configured items, controls changes to characteristics, supports the definitions of baselines, and ensures the integrity of the configuration management data base.

8. Workspace Partitioning. Workspace partitioning consists of supporting multiple projects and multiple teams within a project to provide protected workspaces and common access consistent with the project organization. [8]

9. Access Control. Access control consists of defining types of access rights, implementing control of access to configuration management data base objects based on assigned rights, and controlling the assignment of access rights.

10. Baseline Management. Baseline management consists of identifying a configuration baseline and controlling changes to it.

11. Protection. The protection function consists of ensuring the reliability of the configuration management data base by providing back-up and reconstitution of configuration management information and of preventing the deletion oi needed objects.

12. Status Accounting and Reporting. Configuration status accounting establishes a mechanism for determining how a configuration evolved and where a configuration is at any given time. It provides a means for tracing the history of changes to any configured item. A configured item log which describes the status and functionality of all items under CM should be obtainable.

13 History Reporting. History reporting consists of the reporting of the history of changes to a configuration management data base object. This history should include the date and time of creation of and modifications to the configured item, a description of each change to the configured item, and the name of the individual or organization who authorized and performed the change.

14. Configuration Reporting. Configuration reporting consists of reporting on the content and status of a configuration baseline.

Configuration Management Requirements

The Availability of the Configuration Manager shall be evaluated.

The Capacity of the Configuration Manager shall be evaluated.

The Configuration Management of the Configuration Manager shall be evaluated.

The Costs of the Configuration Manager shall be evaluated.

The Documentation of the Configuration Manager shall be evaluated.

The Extendability of the Configuration Manager shall be evaluated.

The Interfaces of the Configuration Manager shall be evaluated.

The Intraoperability of the Configuration Manager shall be evaluated.

The Maintainability of the Configuration Manager shall be evaluated.

The Efficiency of the Configuration Manager shall be evaluated.

The Rehostability of the Configuration Manager shall be evaluated

The Usability of the Configuration Manager shall be evaluated.

The Proprietary Rights of the Configuration Manager shall be evaluated.

The Granularity of the Configuration Manager shall be evaluated.

The Test Availability of the Configuration Manager shall be evaluated.

The Maturity of the Configuration Manager shall be evaluated.

The Availability of Identification shall be evaluated.

The Configuration Management of the Identification shall be evaluated.

The Costs of Identification shall be evaluated

The Documentation of the Identification shall be evaluated.

The Extendability of Identification shall be evaluated.

The Interfaces of Identification shall be evaluated.

The Availability of Attribute Management shall be evaluated.

The Capacity of the Attribute Management shall be evaluated.

The Extendability of Attribute Management shall be evaluated.

The Interoperability of Attribute Management shall be evaluated.

The Power of Attribute Management shall be evaluated.

The Availability of Version Management shall be evaluated.

The Efficiency of Version Management shall be evaluated.

The Power of Version Management shall be evaluated.

The Availability of Variation Management shall be evaluated.

The Power of Variation Management shall be evaluated.

The Availability of Relationship Management shall be evaluated.

The Capacity of Relationship Management shall be evaluated.

The Completeness of Relationship Management shall be evaluated

The Extendability of Relationship Management shall be evaluated

The Availability of Configuration Control shall be evaluated.

The Configuration Management of Configuration Control shall be evaluated.

The Costs of Configuration Control shall be evaluated.

The Documentation of Configuration Control shall be evaluated.

The Extendability of Configuration Control shall be evaluated.

The Interfaces of Configuration Control shall be evaluated.

The Availability of Workspace Partitioning shall be evaluated.

The Interfaces of Workspace Partitioning shall be evaluated.

The Availability of Access Control shall be evaluated.

The Extendability of Access Control shall be evaluated.

The Power of Access Control shall be evaluated.

The Completeness of Access Control shall be evaluated.

The Availability of Baseline Management shall be evaluated.

The Power of Baseline Management shall be evaluated.

The [illegible]labiliy of Protection shall be evaluated.

The [illegible]ciency of Protection shall be evaluated

The [illegible]stness of Protection shall be evaluated.

The [illegible]lability of Status Accounting and Reporting shall be eval[illegible]d.

The [illegible]lability of History Reporting shall be evaluated.

The [illegible]ndabiliy of History Reporting shall be evaluated.

The Av[illegible]lability of Configuration Reporting shall be evaluated.

4.4 Required Macroscopic Evaluations

4.4.1 Software Development Methodology Support -

Of interest to an organization that is assessing an APSE is the degree to which it supports or excludes the organization's software development methodology. An adequate assessment of an APSE includes the impact it will have on the organization's established procedures and the cost of adapting the APSE to those procedures. To support these concerns, the following evaluations are required.

Requirements

1. Evaluation technology shall be developed to determine the degree to which an APSE supports or excludes specific development methodologies.

2. Evaluation technology shall be developed to determine the degree of flexibility and extensibility of an APSE to support development methodologies.

3. Evaluation technology shall be developed to assess how APSE tools and supported methodologies improve productivity.

4.4.2 Life-cycle Support -

To simplify the APSE concept-of-use, its features should readily relate to users' needs within the framework of the software development life-cycle. The criteria for making this assessment consist of relating each tool to the life-cycle activity it supports, and, for all toolsets, assessing the degree of comprehensive coverage of the life-cycle. Tools that do not apply to a specific activity, and so support multiple activities in a utility sense (editors, report generators, etc.) should be so identified.

Requirements

1. APSE evaluations shall include a classification of APSE tools, which is constructed in a manner to indicate the life-cycle activities addressed by each tool.

2. Evaluation technology shall be developed to assess the degree to which a tool supports the life-cycle activity it addresses.

3. Evaluation technology shall be developed to assess the ease of transition among life-cycle activities

4.4.3 Application Specific Requirements -

An APSE is a collection of software engineering tools. The purpose of building these tools is to employ them in the development of software systems for embedded computer systems. An APSE contributes directly towards addressing the goals of these mission-critical applications. Therefore, it is important to evaluate the capabilities and performance of an APSE specific to the application environment in which it will be employed.

Requirements

1. Evaluation technology shall be developed to assess the capability and performance of the Target System as a result of APSE-produced software.

2. Evaluation technology shall be developed to assess the capability and performance of the host development system and/or the field service system.

3. Evaluation technology shall include application specific benchmarks.

4. Evaluation technology shall be developed to assess distributed and/or multiprocessor systems.

5. Evaluation technology shall be developed to assess simulation support capabilities

6. Evaluation technology shall be developed to assess capabilities for transitioning applications from Host to Target.

7. Evaluations shall include identification and functional elaboration of application specific tools.

4.4.4 Intertool Interfaces -

APSE evaluation results should assist the APSE developer in identifying tool(set)(s) that may easily be integrated into an environment. For example, one may want to integrate a specific vendor's complier into an environment because it meets certain code efficiency requirements. The evaluation of the compiler-to-APSE interfaces should provide information that will assist the APSE developer in determining the effort required to incorpor ,e this compiler into an APSE.

Requirements

1. Evaluation technology shall be developed to assess the degree of coupling between APSE tools.

2. Evaluation of an APSE shall include identification of the APSE toolsets and the tool components that comprize each toolset.

5.0 QUALITY GUIDANCE FOR E&V TECHNOLOGY

<TBD>

6.0 REFERENCES

1. Evaluation and Validation (E & V) Plan, Version 1.0. Air Force Wright Aeronautical Laboratories, Wright-Patterson AFB, 30 November 1983

2. Bailey, E. K., A Framework for Evaluating APSE Useability, 6 June 1984 (Draft).

3. Houghton, R., A Taxonomy of Tool Features for the Ada Programming Support Environment (APSE), NBSIR 82-2625, U.S. Department of Commerce, National Bureau of Standards, December 1982.

4. Probert, T. H., Ada Validation Organization: Policies and Procedures, MTR-82W00103, The MITRE Corporation, June 1982.

5. Nissen, J. C. D., Wichmann, B. A., et al., Ada-Europe Guidelines for Ada Compiler Specification and Selection, October 1982.

6. Castor, V., Criteria for the Evaluation of ROLM Corporation's Ada Work Center, Air Force Wright Aeronautical Laboratories, Wright-Patterson AFB, January 1983

7. Ada Programming Language, ANSI/MIL-STD-1815A, 17 February 1983

8. Orndorff, M. S., Evaluation of Automated Configuration Management Tools in Ada Programming Support Environments (Thesis), AFIT/GCS/EE/84M-1, Air Force Wright Aeronautical Laboratories, Wright-Patterson AFB,March 1984.

9. Proceedings of the Configuration Management Workshop, San Diego, CA, 7-8 June 1983.

10. Buxton, J., Stenning, V., DoD Requirements for Ada Programming Support Environments, "Stoneman", February 1980.

11. Dobbs, P., 400 Generic APSE Questions, Air Force Wright Aeronautical Laboratories, Wright-Patterson AFB,December 1983.

12. Howe,R.G., Evaluation Criteria for Ada Run Time Environments, Mitre Memo D73-M-2415, January 1984.

13. Witte,B., Checklist for Ada math Support Priorities, Ada Letters, April 1984.

14. The DACS Glossary, A Bibliography of Software Engineering Terms, October, 1979.

APPENDIX A

ACRONYM LIST

ACVC Ada Compiler Validation Capability

APSE Ada Programming Support Environment

APSEWG APSE Working Group

CAIS Common APSE Interface Set

CAISWG CAIS Working Group

CM Configuration Management

CVC CAIS Validation Capability

E&V Evaluation and Validation

GFE Government Furnished Equipment

KAPSE Kernel Ada Programming Support Environment

KIT Kernel Interface Team

KITIA Kernel Interface Team Industry/Academic

MAPSE Minimal Ada Programming Support Environment

PUBWG Public Relations Working Group

REQWG Requirements Working Group

RTS Run Time System

TECWG Technical Coordination Working Group

APPENDIX B

COMMAND LANGUAGE INTERPRETER QUESTIONS

Command Language Interpreter Requirements and Questions

Command Language Interpreter/Availability

1. Is the tool completed?

2. If not when will it be available?

Command Language Interpreter/Capacities

1. Do the capacities and limitations of the command language vary from host to host?

2. What is the maximum allowable length of a command stream?

3. What is the maximum number of parameters allowed in a command stream?

4. What is the maximum number of symbols allowed in a command Stream?

5. What is maximum allowed depth of nested substitution?

6. What is the maximum allowed lever of nested loops?

7. What is the maximum allowed level of recursion?

8. Can the limitations and/or capacities be altered without substantially modifying the tool?

9. Can the limitations and/or capactities be altered by the user?

Command Language Interpreter/Configuration Management

1. Is the tool under configuration management?

2. If not, is there an available configuration management organization?

Command Language Interpreter/Costs

COMMAND LANGUAGE INTERPRETER QUESTIONS

1. What is the cost of this tool?

2. Does the vendor retain data rights?

3. What does this cost include?

Command Language Interpreter/Documentation

1. Is the tool documented?

2. What military standards does the documentation meet?

Command Language Interpreter/Extendability

1. Can new features/functions be added to the tool?

2. Can these augmentations be made by the user at will or is vender support required?

Command Language Interpreter/Interfaces

1. Are all appropriate interfaces well defined, documented and as uniform as possible?

2. Does this tool interface with other APSE tools through a common interface?

Command Language Interpreter/Interoperability

1. Can this tool be used with a wide variety of other tools for varying purposes?

2. Is the tool compatible with other APSE's?

3. Can command language sequences be invoked from within Ada programs?

Command Language Interpreter/Maintainability

1. Was this tool built in such a manner that it is maintainable?

Command Language Interpreter/Proprietary

1. Does the vender retain proprietary rights to this tool?

2. Is this tool available GFE?

Command Language Interpreter/Test Availability

1. Are there tests available for this tool?

2. What methods do the tests employ?

Command Language Interpreter/Maturity

1. What is the level of maturity of this tool?

2. How many current users are there?

COMMAND LANGUAGE INTERPRETER QUESTIONS

3. What has been the past usage history?

4. Is the number of users increasing or decreasing?

Command Language/extendability

1. Does the command language allow user defined functions?

2. Does the command language allow tailoring of its functions?

Command Language/Robustness

1. Does the command language contain "do what I mean" features?

2. does the commmand language contain "undo" features?

3. Does the command language contain privileged features?

Command Language/Usability

1. Does the command language contain procedures?

2. What type of support is provided for parameters?

3. What are the scoping rules?

4. Does the command language interpreter allow assignment statements?

5. What types of logical expressions are allowed?

6. What types of arithmetic expressions are allowed?

7. Does the command language allow logical loops?

8. Does the command language allow arithmetic loops?

9. Can the limits in loops depend dynamically on the results of programs initiated by the command language within the loop?

10. What types of conditional control structures are allowed in the command language?

11. Can the outcome of a conditional statement depend dynamically on the results of programs initiated in earlier parts of that conditional statement?

Syntax/Usability

1. Is the syntax of the tabular portions of the command language similar to that of Ada?

2. Is the syntax of the tabular portions of the command language of a uniform nature for the various constructs in the language?

3. By what class of grammar is the tabular portion of the command language representable (regular expressions, linear, LL(1), SLR(O), ...)?

COMMAND LANGUAGE INTERPRETER QUESTIONS

Syntax/Extendabiltiy

1. Must user defined extensions to the command language, if allowed, use the same format as the rest of the command language?

2. Does the syntax make use of the Ada features such as packages and generics which assist in extendability?

Programs/Extendability

1. Does the command language have the cpapbility of defining programs or procedures?

2. Can command language programs contain subprograms?

Programs/Granularity

1. Can any command language sequence be treated as a program?

2. Are self modifying command language programs allowed?

Programs/Interfaces

1. How do command language programs exchange data and/or signals with each other?

2. Can command language programs interact through the APSE database?

Programs/Interoperability

1. Can command language programs be used as parameters to other command language programs?

2. Can Ada programs be used as parameters to command language programs?

3. Are command language programs treated any differently than Ada programs?

Programs/Flexibility

1. Does the command language contain features for encapsulating command language programs?

2. Does the command language contain features for developing generic command language programs?

Tool/Program Invoking Function/Usability

1. What is the nature of the tool/program invoking function?

2. Do the capabilities of the tool/program invoking function differ from that for invoking command language programs?

Tool/Program Invoking Function/Interoperability

COMMAND LANGUAGE INTERPRETER QUESTIONS

1. Does the invocation function differentiate between tool and applications programs?

Diagonistic Generation Function/Usability

1. Does the command language contain provisions for a command language program to generate error messages and/or diagnostics for the user of the program?

2. Does the command language contain facilities for dynamically generating and/or editing diagnostics?

Non-tabular Inputs/Flexibility

1. Does the command language contain provisions for graphical or other non-tabular input?

2. Does the command language contain provisions for creating menu driven command sequences?

3. How is the use of non-tabular inputs integrated with the tabular inputs?

Interpreter/Interoperability

1. Is the interpreter an integrated part of a larger toolset?

Interpreter/Resources Required

1. What hardware resources are required to support the interpreter?

2. What software resources are required to support the interpreter?

Hosts/Availability

1. On what host/operating system combinations is the interpreter available?

2. On what APSEs is the interpreter implemented?

Hosts/Rehostability

1. To what extent does the interpreter use the services provided by the host computer and operating system?

2. Is the APSE on which the interpreter is implemented compatible with KAPSEs implemented on different machines?

Interfaces/Interoperability

1. Does the interpreter have a well defined interface to a database manager?

2. Does the interpreter have a well defined interface to a program library?

COMMAND LANGUAGE INTERPRETER QUESTIONS

3. Does the interpreter have well defined, uniform interfaces to all other programs and APSE tools?

Aids/Usability

1. Does the interpreter provided assistance to the programmer in the form of programming aids such as syntax directed editing, prompters, etc?

2. Can the execution of a command language program be simulated prior to actual execution for debugging purposes?

3 What is the nature of the diagnostics generated by the command language interpreter?

4. Is there a symbolic debugging capbility for command language programs?

5. Are there user setable options within the interpreter?

Per: ...ce/Efficiency

On a host by host basis, what is the expected response time for the interpreter?

APPENDIX C

COMPILER QUESTIONS

Compiler Requirements and Questions

Compiler/Availability

1. On how many hosts is this compiler installed?
   - Approximately how many users have used the compiler?
   - Have there been any major software systems developed using this compiler?

2. What are the target computers for which this compiler can generate code? For each of the targets, approximately how many users are there? To what degree have the targets been exercised? [5]

3. Can the compiler be invoked in both an interactive and a batch mode?

4. Can the compiler be invoked while using other APSE tools? While in the editor, for example.

5. Are there any additional tools supplied with the compiler (e.g., symbolic debugger, target simulator, downloader, linker, etc.)?

6. Are there significant compiler features that could be considered above and beyond that specified in the Ada language specification?

Compiler/Capacity

1. What is the maximum number of users that can invoke the compiler simultaneously? Has this number been verified?

2. What are the limitations on the total (code and data) memory space as occupied by individual packages, tasks and subprograms that can be compiled [12]?

3. What are the limitations on the maximum number of packages, subprograms, or tasks in a compilation?

4. For a small host configuration, are there any restrictions on the size of a program unit that can be compiled? If so, are they indicated in the users' documentation? [5]

Compiler/Configuration Management (CM)

1. What is the general CM plan?

2. Who is responsible for CM (for the supplier)?

3. Are all the source and object modules for a complete version available in one area (e.g., on a tape or a separately controlled disk area)? Is this area accessible to only one person or anyone on the project?

4. How are compiler fixes or enhancements incorporated into a new version?

5. How are new versions controlled and released?

6. If the compiler produces code for more than one target, how are common and machine-dependent modules controlled? For example, are there conventions for naming common function modules for each target?

7. How are approved compiler changes incorporated into new versions? The change approval procedures.

8. Does there exist a list of all modules (including the run time system) that are needed for a complete compiler version? (A version description document).

9. What are the procedures used to update the documentation as a result of compiler changes? Who is responsible to verify that this is done?

Compiler/Cost

1. What are the costs of acquiring the compiler? These costs should be given in terms of:
   - Does the cost include the installation and required maintenance support?
   - Is source code or only object code supplied?
   - Costs of additional (non-supplier developed tools)?
   - Monthly (or periodic) additional maintenance fees?
   - Does the cost include receipt of new versions of the compiler and needed tools?
   - Various cost options depending on licensing (proprietary) arrangements? [5]

2. What is the estimated cost for a compiler rehost?

COMPILER QUESTIONS

3. What is the estimated cost for a compiler retarget? What do the costs in 2 and 3 above include (test, integration, installation, etc.)?

Compiler/Documentation

1. Is a requirements document available? Verify the content and quality.

2. Are complete design specifications available? Verify the content and quality, as follows:
   - An overview of the compiler design showing the major structure and design.
   - Details of the compiler phases and passes.
   - Separate sections which outline the design of the host and each target (to include the run time system(s))
   - Is the design detail sufficient such that an experienced compiler software engineer could maintain the compiler?
   - Are compiler design changes updated in the documents?
   - Is sufficient user documentation available for the host and each target?

3. Does the users (or reference) manual contain an Appendix F which describes all implementation dependent characteristics?

4. Is documentation available for any special tools that were used for the compiler development?

5. Is documentation available for any separate tools that are needed for compiler operation?

Compiler/Efficiency

1. Does the compiler produce assembly language programs, a relocatable object module or interpretive output code? [5]

2. Are object modules interpreted in real time by software (and/or by firmware) in the target computer? [12] Or is there direct machine instruction execution?

3. Is there a facility for management control on the use of low-level (e.g., machine language) features where the use of Ada might otherwise be possible? [5]

4. What is the average number of statements compiled per CPU minute? Are there characteristics of the source text which significantly reduce compiling speed? (Constructs which are likely to influence compiling speeds include the length of the WITH clause, the compilation of a sub-unit, and the presence of generic instantiations.) [5]

5. Are there any alternative structures suggested for the production of low cost, low quality back ends and high quality back ends? [5]

Compiler/Extendability

COMPILER QUESTIONS

1. What were the original design goals of the compiler? [5]
   - Was the design intended for a particular class of users? [5]
   - Are any specific applications envisaged? [5]

2. What high level language(s) is the compiler written in? What percentage is written in assembly language?

3. If written in Ada, were the use of certain language constructs avoided (e.g., tasking, generics, real arithmetic)? [5]

4. Has the compiler successfully recompiled itself?

5. Were any special tools such as a compiler-compiler, translator writing systems, etc., used during the development? If so, are they available to possibly construct additional tools? [5]
   - Do these tools generate a source program of the compiler or do they translate directly into object code? [5]
   - If these tools do not generate an Ada (or other language) program, how can the tools be retargeted? [5]
   - What languages are the tools written in? [5]

Compiler/Granularity

1. What are the major compiler phases? What phases are in memory as the compilation progresses?

2. To what degree are the components of the compiler separable?

3. What parts of the compiler are seen as useful in building other tools? [5]

Compiler/Hardware

1. Are hardware machine dependencies clearly identified in both the code and documentation?

2. Is the compiler designed to use virtual memory? [5]

3. Are hardware dependencies concealed by module interfaces? [5]

Compiler/Interfaces

1. Is the major design interface to a KAPSE or the host operating system?

2. If the interface is to a KAPSE, what KAPSE facilities does the compiler use? [5]

3. Does the compiler operate in a particular APSE? [5] If so, what APSE (or MAPSE) tools does it require, if any? [5]

COMPILER QUESTIONS

4. If not part of an APSE, what characteristics of the host operating system does the compiler rely on? [5] Are all such system dependencies concealed behind module interfaces? [5]

5. Which interfaces are regarded as significant for rehosting or retargeting? [5]

6. What other tools (e.g., symbolic debugger) does the compiler interface with? [5]
   - To what extent are the interfaces documented? [5]
   - Can alternative tools be written conforming to these interfaces?

Compiler/Interoperability

1. What compiler generated information is available to other tools?
   - Symbol table?
   - Cross-reference table?
   - Intermediate forms?
   - Listing outputs?

2. Does the compiler share, or make use of, other APSE (or operating system) tables or information?

Compiler/Maintainability

1. Are instructions available to enable a non-compiler person to install the compiler on an identical host system?

2. Are the procedures for complete compiler generation (from source to executable) documented?

3. What arrangements are available for maintenance? Such arrangements can range from postal service to an on-call maintenance staff. [5]

4. What is the quality of maintenance support?
   - Designated persons for maintenance contact?
   - Availability of maintenance documentation?
   - Telephone query service, visits by supplier staff courses, etc? [5]

5. What are the arrangements for charging for maintenance and/or support of the compiler? [5]

Compiler/Proprietary

1. Can a user install the compiler, or must the supplier do the installation? [5]

2. Are there any proprietary restrictions on compiler release (e.g., no source supplied, data rights, etc.)?

COMPILER QUESTIONS

3. Are there any restrictions on special (non-supplier developed) tools needed for compiler operation? Also, for any optional tools that may be useful?

4. Does the supplier allow others to perform a rehost or retarget? [5]

5. Under what circumstances may the source be made available for a rehost or retarget? [5]

6. What are the licensing arrangements for the compiler (e.g., at how many sites can the compiler be used)? [5]

7. What agreement does the user have to sign before the compiler may be supplied to others? [5]

8. Can a license to distribute the compiler to others be bought or leased? What parts of the compiler (run time system, packages, separate tools, etc.) can be distributed? Can source be included? [5]

9. Can a license to use the compiler be bought outright or leased? [5]

10. What are the arrangements (if any) for the release of information about the compiler's internal structure? [5]

11. Are there any restrictions on the use and/or distribution of software produced by the compiler? It should be noted that the software produced often contains a run time system delivered by the compiler supplier. [5]

Compiler/Rehostability

1. Has the compiler been rehosted?

2. What module (or modules) of the front end (machine independent) need to be modified for the rehost?

3. Is there a manual which describes the steps necessary to rehost the compiler? [5]

4. Are system dependencies adequately isolated and documented? [5]

5. Is there a kit of tools and/or components available to help with the rehosting task? [5]

6. Is the compiler sufficiently modular to allow implementation of critical parts (such as major data structures) to be easily altered for the rehost task? [5]

7. Is an estimate of time given for the rehost?

Compiler/Retargetability

1. What modules of the back end (machine dependent) need to be modified for a retarget?

COMPILER QUESTIONS

2. What modules of the front end need to be modified for a retarget and what are their interfaces?

3. What techniques are used to retarget[5]?

4. Are there any automated tools to aid in the retarget process?

5. Is an estimate of time given for the retarget task?

6. Is there a manual describing the procedures for retargeting? Possibly with examples. [5]

7. Fo. the intermediate language retargeting interface, is the intermediate language tree structured, linear, etc? [5]

8. Is there more than one level of intermediate language at which retargeting is carried out? [5]

9. For the retarget process, what assumptions are made in the design of, and requirements of, the run time system (e.g., tasking monitor, storage allocation scheme, etc.)? [5]

Compiler/Resources Required

1. What is the minimum size of memory needed for one user to run the compiler?
   - Does this change for different targets?
   - Does the size of the source program increase this minimum requirement?
   - As more than one simultaneous user invokes the compiler, by how much does this minimum increase?

2. What is the minimum size of memory needed for one user during each compiler phase?

3. Repeat questions 1 and 2 for secondary storage requirements

4. Are any resources other than primary and secondary storage needed to invoke the compiler?

5. Are compiler phases overlaid to reduce memory occupancy? If so, are any requirements placed on the system? [5]

Compiler/Robustness

1. What safeguards are implemented for protection and recovery against unforeseen system, user and its own failures?
   - Data protection?
   - Internal exception handlers?
   - Trace back facility?

Compiler/Test Availability

1. What tests are available from the supplier to verify compiler operations?
   - Are the tests documented in a test plan?

COMPILER QUESTIONS

- Are instructions for use available?

Compiler/Usability

1. What is the host configuration under which this compiler is installed? [5]

2. What are the host/target configurations under which the produced object code will execute?

3. If the compiler interface is to the host operating system, under which operating system version (or release) does the compiler operate? Also, which target(s) operating system(s)?

4. If the compiler interface is to an APSE, under what version (or release) of the APSE does the compiler operate?

5. What is the character set of the host? [5]

6. What is the character set of non-host targets? [5]

7. Does the users' documentation give helpful user aids on constructs to avoid, features that increase the compiling speed, and features which could aid run time efficiency?

8. Are any phases (especially optimization phases) options selected by either a compile time switch or when configuring the compiler? [5]

Input/Availability

1. Is UNCHECKED-CONVERSION supported? [12] If so, are the details of implementation usage and effects explained?

2. Is PRAGMA-INTERFACE supported? If so, what languages are possible? [12]

Input/Capacity

1. Is there a limit on the number of discrete values for ENUMERATION types? [12]

2. What is the range of values for PREDEFINED INTEGER types? [12]

3. What are the maximum values for REAL types (floating and fixed point)? [12]

4. Are there any restrictions on the use of SHORT variables?

5. For ARRAY types, what restrictions are placed on the number of indices, the range of index values? [12]

6. Is there an upper bound for the number of iterations in a loop statement?

COMPILER QUESTIONS

7. What is the maximum number of nested procedures that are allowed? [12]

8. Is there a limit on the range of literal values which the compiler cannot handle? [5]

9. What is the maximum number of characters permitted on a line of source code? [5]

10. Are there any limits on the number of items of various kinds such as identifiers and strings? [5]

Input/Documentation

1. Is the format of every implementation defined PRAGMA documented? [5,6]

2. Is there a list of all restrictions on representation specifications? [5]

Input/Usability

1. For predefined integer types, what are the values of: [5]

   INTEGER'FIRST — INTEGER'LAST
   SHORT_INTEGER'FIRST — SHORT_INTEGER'LAST
   LONG_INTEGER'FIRST — LONG_INTEGER'LAST
   (Similarly for any other predefined integer types)

2. For floating point types, what are the values of:

   FLOAT'DIGITS — F'MACHINE_MANTISSA
   SHORT_FLOAT'DIGITS — F'MACHINE-EMAX
   F'MACHINE_ROUNDS — F'MACHINE_EMIN
   F'MACHINE_RADIX — F'MACHINE_OVERFLOW [5]

3. In the specification of package SYSTEM, what are the values of [5]: MININT, MAXINT, MAXDIGITS, MAXMANTISSA, FINEDELTA, and TICK.

4. For tasking applications, what are the values of: [5] DURATION'DELTA, DURATION'ACTUALDELTA, DURATION'FIRST, DURATION'LAST, PRIORITY'FIRST, and PRIORITY'LAST

5. What are the values outside the range of safe numbers for real types? [5]

6. Are there any restrictions on the use of the generic procedure UNCHECKEDDEALLOCATION? [5]

7. Are there any restrictions on the use of the generic procedure UNCHECKEDCONVERSION? [5]

Input, Command Language/Availability

1. Does the command language offer the following options:
   - Suppress output of a source listing?
   - Reformat (e.g., indent) the listing? [6].
   - Bring in text (for example, an INCLUDE pragma)? [6]
   - Generate an assembly language listing?

COMPILER QUESTIONS

- Include (or exclude) text in the private part of a package specification?
- Output program statistics?
- Output information relating to compilation phases?
- Output intermediate language forms?
- Suppress certain levels (e.g., informational) of warning messages?
- Suppress certain (or all) optimizations?
- Syntax checking only (no code generation)?
- Output of a cross-reference listing?

Input, Command Language/Documentation

1. Is the design of the command language documented?

2. Are user instructions provided for compiler invocation of each host and target? Do the instructions include:
   - All available options, with defaults and the effects of using certain combinations of options?
   - Use in batch and interactive modes?
   - Invocation of separate compiler tools (if any)?

Input, Command Language/Test Availability

1. Is a series of tests available to verify all command language constructs and options?

Input, User Assistance/Availability

1. Is any on-line user "help" facility available?
   - Is it menu driven?
   - Are there different levels of help (novice and experienced)?
   - Is its use documented in the users' manual?

2. Can the user temporarily halt the compilation at a designated phase and examine the compilation up to that point? [6]. Can the user restart the compilation at the point from which it was temporarily halted? [6]. If so, what are the results and effects (if any)?

3. Can the compilation be immediately halted at a designated phase[6]? If so, what are the results and effects?

4. Can the user request run time tracing for a particular statement or range of statements? [6].

5. Can program flow analysis information be requested? [6] (e.g.,monitoring of frequency of execution of segments of code, timings of segments of code, etc.) [6].

6. Are the effects of executing a source statement which contains certain levels of compilation errors explained in the users' documentation [5]?

Input, User Assistance/Documentation

COMPILER QUESTIONS

1. Is a users' manual available and written at a user's level of understanding? Does it contain:
   - Examples of source files with invocation examples and the resulting compilation listing?
   - Examples showing linking and execution for the host and each target? (Or references which give instructions for these operations.)
   - Explanation of I/O procedures for the host and (in particular) each target?
   - A list, and explanation of, compiler limits, capacities, and restrictions?
   - A list and explanation of each run time system routine (for the host and each target)?
   - Explanations that offer helpful information on run time system options, such as the use (or non use) of certain machine dependencies which might result in more efficient execution or
     - o Any helpful information on preferred Ada programming techniques for the host or target(s)?
     - o A description of parameter passing conventions?
     - o An explanation and examples of invocation of other needed (or optional) tools? [5].
     - o A description and format of parameters and all options which must be supplied to the compiler (or its constituent tools) and the defaults if these are not supplied? [5].
     - o A description of both batch and interactive use along with any differences of results between the two? [5].

Input, Source Statements/Capacity

1. What is the maximum number of source statements that can be compiled?

2. What is the maximum length (in characters) of one source statement?

Input, Source Statements/Documentation

1. Is the permitted character set documented (for the host and each target)? [5].

2. Does the documentation show the allowed placement of every implementation defined PRAGMA? [5].

Input, Source Statements/Usability

1. Are there any limitations or restrictions on the character set that may be used for input source (e.g., substitution characters)?

2. What are the conventions for writing machine code inserts? [12].

Translation/Availability

COMPILER QUESTIONS

1. What are the major phases in the translation process?

Translation/Documentation

1. Does the design documentation give the techniques and algorithms used for translation?
   - Is an overview of the translation process given?
   - Are all of the modules used in the process documented?
   - Is documentation (or references) provided for any special tools that are used?

Translation/Efficiency

1. Is there any possibility of sharing between a global variable and an actual parameter of an array, record, or private type? [12].

2. For type conversions, what is the value of integer X, where X:FLOAT:=1.5? [12].

3. Are there any checks made for possible infinite loops?

4. How is a library unit recognized as a MAIN program? [12].

5. How is a library unit initiated? [12].

6. What parameterization is allowed for a library unit (e.g., can it be a function), and if so, how are parameters received and delivered? [12].

7. Are there any restrictions on the form of, or statements contained in, a MAIN program, as opposed to its subunits? [12].

8. Under what circumstances will code or read-only data be shared between different instantiations of some generic unit, and what control (if any) can the user exercise over this? [12].

9. What is the effect of using uninitialized variables? Does the compiler flag or reject a program that depends upon such variables[5]?

10. What is the interpretation of expressions that appear in address specifications, including those for interrupts? [5].

11. What conventions are used for any system-generated names denoting system-dependent components? [5].

12. Are compilation units (packages) automatically recompiled (if necessary) in order to properly compile another compilation unit? [5].

13. When a program is recompiled, is any use made of the previous compilation to increase compilation speed? [5].

14. If a package specification is changed by adding a declaration, does the recompilation make units which used this package obsolete. (The compiler could optimize this

case by noting recompilation.)

15. Are static (sub)expressions always evaluated (even when not required by the language reference manual)? [5].

16. With the statement: A(I): = A(I)+1; is the address of A(I) evaluated once or twice? Also, is a special instruction generated[5]?

17. With matrix computations:

```
for I in 1 .. N loop
    for J in 1 .. M loop
        A(I,J) ...
    end loop;
end loop;
```

    Is the address of A(I,J) calculated each time by multiplication? Does hardware do this anyway? Or does the compiler generate increments through the array (strength reduction)? [5].

18. Assuming N and M above are constants (but not literal values), is index-bound checking performed when A'FIRST(1)=1, A'LAST(1)=N, etc [5]?

19. Is there a facility for providing a fully qualified name (e.g., the result of the overloading resolution)? [5].

20. What is the effect of undeclared identifiers? [5].

Translation/Extendability

1. What special tools were used for implementation of the translation phase? For example, a parser generator, lexical analyzer, etc. In what language(s) are they written?

Translation/Interfaces

1. Does the design documentation give the interfaces for the translation modules?

Translation/Interoperability

1. Is any information resulting from the translation process useful to other APSE tools? Is it stored in a separate file for subsequent use?

Translation/Maintainability

1. Is the code and documentation of sufficient quality to perform maintenance?

2. If non-implementor developed tools are used, is maintenance of these possible or available?

Translation/Proprietary

1. Are there any cost or proprietary restrictions on the tools used in the translation process?

Translation/Rehostability

COMPILER QUESTIONS

1. Are all of the translation modules machine independent, or is some modification needed for a rehost?

2. If the translation tools or modules are not written in Ada, are they available on the new host?

Translation/Resources Required

1. For each translation phase: What is the memory requirement? What is the secondary storage requirement?

Translation/Robustness

1. What techniques are used to handle unforeseen error situations during the translation process? Special exception handlers, etc?

Translation/Test Availability

1. What internal compiler tests are available to verify the correctness of the translation phases?

Translation, Analysis/Availability

1. In addition to Ada language requirements analysis, are any other forms of analysis performed? Statistical analysis? Program flow analysis?

2. Are there varying levels of severity for compiler-generated diagnostics? If so, what are the levels of severity and what distinguishes one from another? [6].

Translation, Analysis/Documentation

1. Is design documentation available on the various forms of analysis that are performed?

Translation, Analysis/Efficiency

1. What is the overhead during compilation for the various analysis options?

2. What steps are taken to avoid "cascading" of compilation errors? [5].

Translation, Analysis/Granularity

1. Are any of the analysis modules useful as separate tools?

Translation, Analysis/Interoperability

1. Is the analysis information available to other tools? Is it stored on separate files? Is it in human-readable form?

Translation, Analysis/Proprietary

1. Are there any costs or proprietary restrictions on any of the tools used for analysis functions?

Translation, Analysis/Test Availability

COMPILER QUESTIONS

1. Have all compiler diagnostics been verified to insure that the text is inconsistent with the actual error?

Translation, Analysis/Usability

1. Is the use of the analysis facilities (with options) described in the users' manual? Is the interpretation of the information described?

Translation, Intermediate Forms/Availability

1. What are the intermediate representations of the source program during the translation phases?

2. Are any standard intermediate languages (such as DIANA) used? [5].

Translation, Intermediate Forms/Documentation

1. Are the intermediate forms documented? The design and implementation for each translation phase should be given. Is the detail and quality sufficient to permit understanding?

Translation, Intermediate Forms/Extendability

1. In what language(s) are the modules that produce the intermediate forms written?

2. Are any non-implementor developed tools used to produce or interpret the intermediate form?

Translation, Intermediate Forms/Interfaces

1. What is the interface structure of the intermediate forms? [5].

2. Are the input and output interfaces for the modules that process intermediate forms explained?

Translation, Intermediate Forms/Interoperability

1. Are any of the intermediate forms useful to other tools, such as a statistical analyzer, etc?

Translation, Intermediate Forms/Proprietary

1. Are there any costs or proprietary restrictions on any of the tools used for intermediate forms development and processing?

Translation, Intermediate Forms/Rehostability

1. If the modules used to produce the intermediate forms are not written in Ada, can they be made available on another host?

Translation, Intermediate Forms/Test Availability

1. Are any internal tests available to verify the correct intermediate form sequences?

Translation, Intermediate Forms, Usability

COMPILER QUESTIONS

1. Are any intermediate forms results accessible to the user (as an option) [5]. Is there a human-readable form? [5]. Can the representation be written to a file for subsequent use?

Translation, Optimization/Availability

1. What optimizations are performed in the front end? [5].

2. Can programs be optimized via code sharing of generic units, merging of compilation units, etc? [12]

Translation, Optimization/Documentation

1. Does the design documentation describe the algorithms and techniques used for optimization implementation?

Translation, Optimization/Extendability

1. Are any additional optimizations planned?

   Are any non-implementor developed tools used for optimization?

Translation, Optimization/Rehostability

1. If the modules that perform optimization are not written in Ada, can they be made available on the new host?

Translation, Optimization/Test Availability

1. Have tests (benchmarks) been run to verify the memory (or efficiency) gained with and without each of the optimization options? [6].

Translation, Optimization/Usability

1. In addition to the PRAGMA OPTIMIZE (Annex B, [7]), are there any additional options to select or turn off other optimizations?

2. Can a user specify levels of optimization for memory and/or efficiency as an option (e.g., additional PRAGMAs)? [6].

Translation, Symbol Table/Capacity

1. For each symbol table entry, list the user capacity that is available (if applicable) (e.g., maximum number of identifiers). Can any of these limits be set by the installation or user?

Translation, Symbol Table/Documentation

1. Is the structure of the symbol table design described?

2. Is each symbol table entry (size and type) described?

3. Are the techniques used to build the symbol table described?

Translation, Symbol Table/Efficiency

COMPILER QUESTIONS

1. During compilation, is the symbol table kept in memory or on secondary storage? Or does this depend on the size of the compilation unit?

2. During the compilation process, how are symbol table entries accessed?

Translation, Symbol Table/Interoperability

1. Is any of the symbol table information used (or useful) by other non-compiler tools?

2. Can other tools access the internal symbol table information?

3. Is any symbol table information stored in separate files for subsequent use (possible in a reformatted form)?

Translation, Symbol Table/Resources Required

1. What is the minimum size of memory and secondary storage needed for the symbol table for one compilation unit?

Translation, Symbol Table/Test Availability

1. Are there any internal tests available to verify the correctness of the symbol table entries?

Translation, Symbol Table/Usability

1. Are there any user options for access of, or use of, the symbol table information? If so, are they described in the users' manual?

Code Generation/Availability

1. Are any target code generators available other than the host?

2. Is PRAGMA SUPPRESS supported? [12].

3. Is PRAGMA ELABORATE supported for user library units? [12].

Code Generation/Capacity

1. What is the limit on the size of the object module that can be generated?

Code Generation/Documentation

1. Is there a list of generated code sequences for each language production?

2. Is the design of the various methods of passing parameters documented [5]?

3. Is the mechanism for returning results from a subprogram given (especially where the result is a record or an unconstrained array type) [5]?

4. Is the implementation of generic subprogram parameters described [5]?

Code Generation/Efficiency

1. What is the general methodology or techniques used for code generation?

2. What procedures were used to insure that the most efficient code sequences were selected for each language construct (e.g., expert knowledge of the machine instruction set)?

3. What techniques are used in range and constraint checking? [5].

4. Is space allocated for variables declared but not used? [5]

5. What is the general overhead for procedure calls and returns?

6. How is PRAGMA INLINE treated? [5].

7. What are the instruction sequences for procedure calls and returns?

8. How many instructions are generated at scope entry for exception handling? [5].

9. If the only generic parameter is a subprogram, is the executable code duplicated for each instantiation of the generic code package (subprogram)? [5].

10. If a machine has only (say) two predefined integer types, are just two copies made of a generic package/subprogram with the sole parameter? The same question for floating types. [5]

11. Are any additional object code or data requirements imposed by the use of generics? (Especially when code sharing is in use?) [5].

12. With a paramaterless generic package, is the executable code duplicated for each instantiation? [5].

13. Under what circumstances are parameters passed by reference or copy [5]?

14. For scaler types, are range checks applied to every subexpression or only at final assignment? [12].

15. Are the following forms of minimizing constraint checking performed [5]?

```
I:  INTEGER range -2 .. 2;
J:  INTEGER range 0 .. 10;
type AT is access T;
V:  AT;
I:=  22 mod 3; -- (1) no checks needed
I:= J;         -- (2) check on top limit only
V:=  new T(...);
if V.L = ... then -- (3) no null access check
                  -- (4) current variant is correct
```

COMPILER QUESTIONS

16. Are subprograms which are declared in a package but not used loaded into the program? [5].

17. If a subroutine is only called once, is a subroutine call, return sequence generated, or is the subroutine code just planted in line [5]?

18. What are the techniques of implementation of PRAGMA SUPPRESS?

19. Does the compiler generate code from source text which has resulted in compiler-generated warnings? If so, is an option provided to enable the user to suppress code generation in such cases? [6].

20. What precision and ranges of accuracy are supported for SHORTFLOAT and LONGFLOAT types? [12].

21. For fixed point types, what is the range and accuracy of ACTUALDELTA? [12].

22. Is there a limit on the accuracy of real literal expressions? [5].

23. What is the approximate ratio of generated machine instructions to an Ada statement?

Code Generation/Granularity

1. Are any of the code generation components useful as separate tools?

Code Generation/Hardware

1. Does the code generator take advantage of special host or target features such as address calculations?

Code Generation/Interfaces

1. Is the level of interface documentation (in the code and specifications) sufficient?

Code Generation/Interoperability

1. In addition to the object module, is there any information produced by the code generation phase that is useful to other APSE tools?

Code Generation/Maintainability

1. Are there any special tools (non-implementor developed) that are used for code generation? If so, is maintenance available?

Code Generation/Proprietary

1. Are there any proprietary or data rights restrictions on the use of, or distribution of, any of the modules or tools used in the code generation process?

Code Generation/Rehostability

COMPILER QUESTIONS

1. Which code generation modules need to be modified for a rehost? Are host dependencies given in the documentation and code?

Code Generation/Retargetability

1. Which code generation modules need to be modified for a retarget? Are target dependencies given in the code and documentation?

Code Generation/Test Availability

1. What tests or techniques are used to verify correct code sequences?

2. Are there any tests (benchmarks) that are used to determine object code efficiency against identical hand-coded assembly language programs: If so: Are the tests available to users? What are the results?

Code Generation/Usability

1. Are there any circumstances under which it is possible to share code between two different generic instantiations (including any user control that is available)? [5].

2. Is there any method of relating the object code to the source program? [5].

Code Generation, Debugging/Availability

1. Are user debugging facilities provided? If so:
   - What functions are implemented?
   - Is the debugging at the high-level language level or at the machine level?
   - Are any separate target-specific debugging facilities provided?
   - Are any system debugging facilities available through the compiler? (These are generally at the machine level.)

Code Generation, Debugging/Documentation

1. Is the design of the debugging facilities documented? Is this documentation a part of the compiler design specifications or are they independent documents?

Code Generation, Debugging/Efficiency

1. In what ways might optimization adversely affect the use of the symbolic debugger on the target machine?

2. For each separate debugging function, what is the compile time and run time overhead? For example, what is the ratio for run time with and without this debugging feature?

3. For each separate debugging function, what are the additional memory requirements?

COMPILER QUESTIONS

4. What are the implementation techniques for the debugging functions? For example, at the start of a statement machine code sequence, is code inserted to branch to a debugging module?

Code Generation, Debugging/Granularity

1. Are the debugging facilities separate tools or is the code built into the compiler?

Code Generation, Debugging/Hardware

1. Are there any special hardware dependencies for the debugging functions?

Code Generation, Debugging/Interfaces

1. What special compiler interfaces are needed for the debugging functions?

   If the compiler and debugging facilities are separately developed, is there a formal interface description between the two?

Code Generation, Debugging/Interoperability

1. Are any of the results of the debugging capabilities available and/or used by other APSE tools? For example, statistical analysis functions.

Code Generation, Debugging/Proprietary

1. Are any of the debugging facilities proprietary? For example, supplied by the system vendor?

Code Generation, Debugging/Rehostability

1. Which debugging modules need to be modified for a rehost?

Code Generation, Debugging/Retargetability

1. Which debugging modules need to be modified for a retarget?

Code Generation, Debugging/Resources Required

1. In addition to memory, are there any other resources needed to use the debugging functions (disk, tape, etc.)?

Code Generation, Debugging/Test Availability

1. Are there any internal compiler tests to verify the correctness of the debugging functions? If so, is their use explained?

Code Generation, Debugging/Usability

1. Is the use of the debugging facilities adequately explained?

2. Is any on-line assistance for the debugging functions available?

Code Generation, Optimization/Availablity

1. Are there any target-specific optimizations implemented?

2. At what stages in the generation of object code can target-specific optimizations be made? [5]. Can additional optimization stages be incorporated, either at the outset or later if proved necessary? [5].

3. Are there any standard tools, algorithms or components to aid in target-specific optimizations? [5].

Code Generation, Optimization/Documentation

1. Are the target-specific optimizations documented in both the design and users' documentation at the appropriate level of understanding? Are implementation techniques explained?

Code Generation, Optimization/Efficiency

1 How, if at all, does target-specific optimization interact with any front-end optimization? [5].

2. Is dead code eliminated from if and case statements? If so, is the user notified? [5].

Code Generation, Optimization/Hardware

1. Are there any target-specific optimizations that take advantage of hardware features and efficiencies?

Code Generation, Optimization/Proprietary

1. Are any of the target-specific optimization modules proprietary?

Code Generation, Optimization/Retargetability

1. Which target optimization modules need to be modified for a retarget?

Code Generation, Optimization/Test Availability

1. Are tests available to verify the efficiency gained with target-specific optimizations? If so, is their use and the interpretation of results explained?

Code Generation, Optimization/Usability

1. Are any of the target optimizations user selectable as options?

2. Does the users' documentation give any helpful information on preferred coding techniques to take advantage of compiler target optimizations?

Output/Availability

COMPILER QUESTIONS

1. Is there a tool that shows the structural relationship between the modules of a compiled system? [5].

2. Is pretty printing of Ada source text possible?

3. Is reformatting of Ada source test possible? [6]. Can user-supplied packages replace or provide the facilities of Questions 2 and 3? [5].

4. Can additional output options be provided by the user? [5].

5. Can the compiler generate a history file which records who performed the compilation and when it was performed? [6].

6. Does the compiler generate a file which documents the modules and associated revision levels which were used to produce a particular configuration? [6].

Output/Documentation

1. Does the design documentation give the implementation techniques of the various compiler outlets? This should show the data structures, as well as the modules that perform the outputs. Included should be outputs that are used for internal compiler use (e.g., compiler debugging and maintenance).

2. Does the users' documentation explain the compiler outputs:
   - Examples?
   - Instructions for use and interpretation?
   - Various user options?

Output/Hardware

1. Do any of the compiler-generated outputs depend on specific hardware peripheral devices?

Output/Interfaces

1. Are output interfaces documented in both the design documentation and the code?

Output/Resources Required

1. Does the design and users' documentation give an estimate of resources (in particular, secondary storage) for any special outputs (e.g., statistics, flow analysis, trace analysis, etc.)? The user should be cautioned on outputs which may generate large volumes of data.

Output/Test Availability

1. What internal compiler tests are available to verify correctness of outputs?

Output/Usability

COMPILER QUESTIONS

1. Can error messages be redirected to a user-defined file? [12].

2. Is there a facility which allows association (or link) of a source listing with the latest object module? [5].

3. During interactive use, are error messages displayed to the user as the compilation proceeds?

4. Is the output of intermediate language forms reducable and usable?

Output, Analysis/Availability

1. In addition to source program error analysis, what other forms of analysis are available for user output:
   - Program flow analysis?
   - Statistical analysis?
   - others?

Output, Analysis/Documentation

1. Is the overall structure of error and warning messages documented? [5]. e.g., Fatal, Serious, Recoverable, Warning, Informational, etc. (The documentation should show the message number and the text that is output for each error).

Output, Analysis/Extendability

1. Can any user packages be substituted for compiler-implemented analysis outputs: Different output formats? Redirect to other devices?

Output, Analysis/Granularity

1. Are any of the analysis output modules useful as separate tools?

Output, Analysis/Interfaces

1. Are the interfaces of the analysis tools documented in the design documentation?

Output, Analysis/Interoperability

1. Are any of the analysis outputs or information useful to other APSE tools?

Output, Analysis/Rehostability

1. Are any machine dependencies for modules that process and produce analysis outputs documented in the design documentation?

Output, Analysis/Test Availability

COMPILER QUESTIONS

1. Are internal tests available to verify correctness of analysis functions and outputs?

Output, Analysis/Usability

1. Following are warning messages that are considered to be useful to the user. Are they given?
   - A statement whose static properties guarantee that an exception will be raised. [5].
   - An unusually expensive construct. [5].
   - A real expression whose accuracy is inherently low. [5].
   - Declared identifiers that are not used. [5].
   - Elimination of unreachable code. [5].
   - Declared packages that are not invoked. [5].
   - Erroneous or poor programming practices. [5].
   - Identifiers used but not initialized. [5].
   - Infinite loops. [5].
   - Ignored (unimplemented PRAGMAs) [5].
   - Code motion affecting debugging. [5].

2. Is any user information provided which gives the user a restart position to allow recovery from compilation errors. [5].

3. Is any information output to aid the user in resolving ambiguous overloadings? [5].

Output, Cross-Reference/Availability

1. Is there an option to obtain a cross-reference listing?

Output, Cross-Reference/Documentation

1. Is the format, design and implementation techniques for the cross-reference information described in the design documentation?

Output, Cross-Reference/Test Availability

1. Are internal tests available to verify the correctness of the cross-reference information and output?

Output, Cross-Reference/Usability

1. Is the use and interpretation of the cross-reference listing explained in the users' documentation?

2. Does the cross-reference listing include:
   - Alphabetical list of identifiers with statement numbers of where declared, initialized and used? All identifiers' types and attributes?
   - Does the output indicate if the identifiers were brought in from a package?

3. Are there any user options when selecting a cross-reference output?

Output, Listing/Documentation

COMPILER QUESTIONS

1. Is a list of all outputs, options and defaults for the listing given in the users' documentation:
   - Examples?
   - Instructions for use?
   - Instructions for interpretation?

Output, Listing/Test Availability

1. Are internal tests available to verify the correctness of listing outputs?

Output, Listing/Usability

1. Does the listing include:
   - The source program with statement numbers?
   - The total memory space of the module?
   - The CPU compilation time?
   - The compiler version number and date of release?
   - An image of the compiler invocation commands?
   - A list of internals, externals and packages (including run-time system modules)?
   - Compilation units (packages) made obsolete by the compilation of a unit? [5].
   - Any compilation units that were automatically recompiled in order to properly compile another compilation unit? [5].

2. Is an optional assembly language listing available? Does it include:
   - Generated machine instructions (opcode, operands, appropriate comments, etc.)?
   - The compiler-generated statement number for the first machine instruction of the sequence?
   - Calls made to run-time system routines? [6].
   - Calls made to system routines?
   - Assembler mnemonics?

3. Is the statistics information readable and useful?

4. Does the listing state that certain levels of errors or warnings were suppressed (as selected by the user)?

5. Do error messages indicate the statement number and column of the error (or other suitable identification)?

6. Is any information (other than the error message) provided to help the user find the cause of the error?

7. Does the listing give optimizations that were used and/or suppressed by the user?

Output, Object Module/Documentation

1. For the host and each target:
   - Is the design of the object module documented (implementation techniques and format)?

COMPILER QUESTIONS

- Is the format of internal references from one compiled unit to another given? [5].
- Does the design and users' documentation give the conditions under which an object module will or will not be produced (e.g., certain levels of errors)?

Output, Object Module/Interfaces

1. How are interfaces between the object module and the next step towards execution handled? Passed directly to the host or target linker? A Linker/Loader?

2. Is there a document (e.g., interface control document) which fully describes the interfaces?

Output, Object Module/Rehostability

1. Which modules that process or produce the object module needs to be modified for a rehost?

Output, Object Module/Retargetability

1. Which modules that process or produce the object module need to be modified for a retarget?

Output, Object Module/Test Availability

1. Are there any internal tests available to verify the correctness of the object module?

Output, Object Module/Usability

1. Is there an option which allows suppression of object module generation upon occurrence of certain levels of errors?

2. Does the compiler produce an object module even if certain errors are present? If so, under what circumstances? [5].

3. Is the format and content of the object module given in the users' documentation? Are references or explanations given for linker information (e.g., link item types)? This is needed for user understanding and debugging.

4. How is downloading from the host to the target accomplished? [12].

5. Are object modules linked by the host (or target) linker, or is an an APSE linker required? [5].

RTS/Availability

1. Does the RTS depend on the host (or target) operating system functions, or could it operate on a "bare" machine?

2. Are system initialization (startup) functions provided? [12].

3. What values are supplied in declarations in PACKAGE SYSTEM for the following: ADDRESS, NAME, MEMORY, MAXDIGITS, MAXMANTISSA? [12].

4. Are there any facilities for performance measurements for CPU time utilization, dynamic memory utilization, I/O channel utilization, etc? [12].

5. What system status information is accessible to Ada programs from the run-time environment? [12].

RTS/Documentation

1. Is a detailed description of the design of each RTS module given for the host and each target?

2. Is any information given to allow modification of the RTS to provide better efficiency, performance, etc., for different applications?

3. Does the documentation give the source language that each module is written in?

4. If any of the RTS modules are non-implementor developed, is documentation available?

RTS/Efficiency

1. Are RTS modules that are not needed for a program still included in the object module?

2. Does the implementation use Julian dates versus calendar dates? [12].

3. For clock initialization/synchronization, what are the values of FINEDELTA and TICK? [12].

4. How much CPU time is expended while accessing the system clock? [12].

5. Does the initialization routine free itself up to become overlayed with working storage? [12].

6. How is initiation and communication with a main program handled? Are there any restrictions? [5].

RTS/Extendability

1. In what programming language(s) is the RTS written? [12].

2. If not written in Ada, are there plans to translate the RTS to Ada?

3. What percentage of the RTS is in assembler language?

4. Are there any standard components available to help in the construction of the RTS (e.g., tasking monitor written in Ada)? [5].

RTS/Granularity

COMPILER QUESTIONS

1. Are any RTS modules separately selectable and useful as other tools?

RTS/Hardware

1. Are any hardware diagnostics (e.g., memory tests) performed? [12].

2. Can a user effect orderly shutdown of the system? [12].

RTS/Interfaces

1. Do the compiler-invoked RTS modules interface with the host operating system or the KAPSE?

2. Are the interfaces for each RTS module documented?

RTS/Maintainability

1. If RTS modules are supplied by other than the implementor, is sufficient maintenance documentation available? Or must the supplier perform maintenance?

RTS/Proprietary

1. Are source listings of the RTS available? What proprietary rights are involved? [12].

2. Are there any proprietary rights to any of the RTS modules?

RTS/Rehostability

1. Which RTS modules need to be modified for a rehost? Are instructions available?

RTS/Retargetability

1. Which RTS modules need to be modified for a retarget? Are instructions available?

2. For a retarget, does the presence of other tools in the compilation system or APSE affect the back end [5]?

RTS/Resources Required

1. What is the total memory space of the entire RTS?

2. What is the memory space of each RTS module?

RTS/Robustness

1. What general forms of recovery are supported for machine, system or process failure?

RTS/Test Availability

1. What internal tests are available to verify the correctness of the RTS modules?

RTS/Usability

COMPILER QUESTIONS

1. Can a user access the RTS (operating system) directly? [12].

2. Can a user substitute their own RTS packages for compiler-supplied packages? If so, are instructions available?

RTS, Memory Management/Availability

1. Does the implementation support both primary and secondary stack management? [12].

2. What garbage collection capabilities are available for deallocating working storage? [12].

3. Does the RTS support memory partitions, overlays, swapping and program segmentation? [12].

4. Is the use of virtual memory supported? [12].

5. Does the implementation provide for memory protection for both code and data? [12].

6. Does the implementation support the use of checkpointing (e.g., saving the state of the system)? [12].

RTS, Memory Management/Capacity

1. What are the minimum and maximum allocations of working storage that can be made for individual tasks, subprograms and packages? [12].

2. Are there any restrictions on the maximum memory space for ACCESS types? [12].

RTS, Memory Management/Documentation

1. Are the RTS memory management algorithms and techniques documented in the design documentation?

2. Does the documentation give references to system documentation relating to memory management (if applicable)?

3. Is a description of primary stack management given? [5].

4. Is a description of secondary stack management (if any) given? [5].

5. Is the method of acquiring and releasing space for tasks documented? [5].

6. Is the method of ACCESS type collection management documented? [5].

7. Is a description of heap management and garbage collection given? [5].

RTS, Memory Management/Efficiency

COMPILER QUESTIONS

1. What are the techniques and algorithms for allocating working storage? [12].

2. Is deallocation of working storage automatic in all cases? When does deallocation occur? [12].

3. What techniques are used to determine the amount of unused working storage still remaining to be assigned? [12].

4. Is working storage allocated in blocks which are (or can be) of unusual length? [12].

5. What techniques are available for detecting fragmentation problems in the working storage area? [12].

6. Is working storage periodically compacted or coalesced?

7. If paging or segmentation is supported, how are thrashing problems dealt with? [12].

8. What type of control is afforded by the LENGTH CLAUSE? [12].

9. What allocation scheme is used for ACCESS types (e.g., stack, dynamic, fixed, etc.)? [12].

10. For record representation clauses, will objects in a collection always have the same length if the designated type is an "Unconstrained ARRAY" or an "Unconstrained RECORD" type with discriminants? [12]. Are any restrictions placed on the use of alignments? [12].

RTS, Memory Management/Hardware

1. What hardware-dependent features are used for memory management (e.g., protection registers, virtual address translation mapping registers, etc.)?

RTS, Memory Management/Interfaces

1. Do the RTS memory management modules use (or interface with) any system-supplied software?

RTS, Memory Management/Test Availability

1. What internal tests are available to verify the correctness of the memory management functions?

RTS, Memory Management/Usability

1. May a user statistically assign an amount of workng storage after compilation, but before run time? [12].

2. May a user optionally specify memory protection for data segments?

RTS, Task Management/Availability

1. Is tasking performed as part of the compiler RTS or is it performed by the host (or target) operating system?

2. Are priorities used for the following operations: Queuing for rendezvous, queuing for real memory, task start-up elaboration, raising exceptions, terminations, input and output? [12].

3. Is expedited dispatching supported? [12].

4. Is the use of privileged tasks supported? If so, are there several levels of privileged states? [12].

RTS, Task Management/Capacity

1. What are the maximum lengths for the TASK queue and the DELAY queue? [12].

RTS, Task Management/Documentation

1. Is the design of the task management functions given?

RTS, Task Management/Efficiency

1. What range of context switch times is considered to be typical? [12].

2. What is the maximum time duration in which all interrupts may be inhibited? [12].

3. Are all tasks in a library package required to "terminate" upon completion? [12].

4. What scheduling disciplines are used (e.g., FIFO, round robin (with time slicing), etc.)? [12].

5. What restrictions are placed on the range of ENTRY families? [12].

6. What overhead times are incurred while creating, interrupting, terminating and/or aborting a task? [5].

7. What are the circumstances for a reschedule (and its duration)? [5].

8. In a message-based system, how are intermediate tasks ("agents," "messengers") compiled? In many cases, such tasks can be removed. Is such optimization handled? [5].

9. How is stack and heap space acquired for a new task in a multi-tasking program? [12].

10. What is the method of implementing the Ada rendezvous mechanism? For example, is an Ada-run time kernel used? Or does the implementation use the target operating system facilities? [5].

11. What is the method of passing parameters in a rendezvous? [5].

12. In a rendezvous, is the rendezvous code executed by the owning task, or can it be executed by the calling task? [5].

13. For a DELAY statement, what is the RANGE and DELTA for DURATION? [12].

14. What is the accuracy of the DELAY statement? [5].

15. What is the overhead (time) to access the clock in DELAY statements? [5].

16. What is the accuracy of the real-time clock? [5].

17. What is the method of associating internal interrupts with task entries? [5].

18. Are there any implementation task management optimizations that have been achieved? [5].

19. How much space is required for a general and for a passive task? [5].

RTS, Task Management/Interfaces

1. What is the interaction between task management and exception handling? [5].

RTS, Task Management/Robustness

1. What capabilities are available for detection of CPU overload conditions? [12].

2. What capabilities are available for detection of CPU deadlock conditions? [12].

3. What capabilities are available for detection of CPU indefinite postponement conditions?

RTS, Task Management/Test Availability

1. What internal tests are available to verify the correctness of task management functions?

RTS, Task Management/Usability

1. Does the implementation maintain an audit trail (e.g., in an end- around buffer) giving the order in which tasks are initiated, suspended, terminated and/or aborted? [12].
   - How does the user obtain this information for real-time recording, and/or off-line diagnosis (e.g., following crashes)? [12].
   - Are entries time tagged? If so, what is the granularity of time? [12].
   - Do these functions result in real-time processing overhead, or are they performed parallel to the task processing?

RTS, Task Management, Distributed Processing/Availability

1. Is distributed processing supported? If so, what is the general implementation technique (e.g., master/slave, symmetrical, etc.)?

2. Is rendezvous supported in distributed processing? [12].

3. Are procedure calls to a remote processor supported? [12].

RTS, Task Management, Distributed Processing/Capacity

1. How many distributed processors are supported?

2. What is the limit on the amount of information which can be exchanged between tasks during a rendezvous? [12].

RTS, Task Management, Distributed Processing/Documentation

1. Are the design and implementation techniques for distributed processing documented?

RTS, Task Management, Distributed Processing/Efficiency

1. What typical time delays (overhead) are there for rendezvous between processors? [12].

2. In which processor(s) do the RTS modules reside?

3. What support is available for clock synchronization between processors? [12].

4. How are task termination dependencies enforced? [12].

5. How are tasks initiated in another processor? [12].

RTS, Task Management, Distributed Processing/Hardware

1. Does data transfer between processors occur via calls or through high-speed memory? [12].

RTS, Task Management, Distributed Processing/Resources Required

1. What is the overhead for the distributed processing functions in terms of memory requirements?

RTS, Task Management, Distributed Processing/Robustness

1. What internal safeguards have been implemented to detect and recover from own or system errors?

RTS, Task Management, Distributed Processing/Test Availability

1. What internal tests are available to verify the correctness and efficiency of the distributed processing functions? Have the tests been executed using more than one processor? Are test results available, or can they be run by the user?

RTS, Task Management, Parallel Processing/Availability

1. Is parallel processing supported by the RTS? If so, what are the general implementation techniques? How many parallel processors are supported?

RTS, Task Management, Parallel Processing/Capacity

COMPILER QUESTIONS

1. What are the maximum number of tasks that can execute in parallel? Has this number been verified?

RTS, Task Management, Parallel Processing/Documentation

1. Is the design of the algorithms and techniques of the parallel processing modules documented?

RTS, Task Management, Parallel Processing/Rehostability

1. Which parallel processing modules need to be modified for a rehost?

RTS, Task Management, Parallel Processing/Retargetability

1. Which parallel processing modules need to be modified for a retarget?

RTS, Task Management, Parallel Processing/Test Availability

1. [illegible] tests available to verify parallel processing operations? Do these tests simulate, or have they been executed with tasks running in parallel on more than one processor? On how many processors have the tests been run? Are test results available?

RTS, Exception Handling/Availability

1. What types of recovery from either machine failure or process failure are supported? [12].

2. Does the implementation support the use of PRAGMA SUPPRESS for the following CHECKs: ACCESS, RANGE, DISCRIMINATE, INDEX, LENGTH, DIVISION, OVERFLOW, ELABORATION, STORAGE? [12]. If so, how are exceptions handled?

RTS, Exception Handling/Documentation

1. Does the design describe the way in which exception numbers are allocated? [5].

2. Is the mechanism used to bind a raised exception to the appropriate handler described [5]?

RTS, Exception Handling/Efficiency

1. What are the techniques and overhead associated with exception handling? [12].

2. How are "orphan" processes treated during exception handling? [12].

3. Under which conditions are the exceptions NUMERICERROR, PROGRAMERROR OR STORAGEERROR raised? [5].

RTS, Exception Handling/Hardware

COMPILER QUESTIONS

1. What are the interactions between exception handling and the host or target hardware? [5].

RTS, Exception Handling/Interfaces

1. What are the interactions between exception handling and the host or target operating system? [5].

RTS, Exception Handling/Test Availability

1. Are internal tests available that force an exception to invoke each exception handler?

RTS, Exception Handling/Usability

1. Is exception handling described in the users' documentation?

2. Are any user options regarding exception handling given in users' instructions?

RTS, Data Management/Availability

1. Is machine I/O implemented as part of the RTS?

2. What low-level I/O drivers are supported? [12].

3. Is asynchronous I/O supported for character and block-oriented devices? [12].

4. Is there text I/O support for variable spacing margins, page numbering, headers and footers? [12].

5. Is formatted I/O (analogous to COBOL or FORTRAN) supported? [12].

6. Does the implementation use backing storage for I/O (e.g., spooling)? [12].

7. What type of support is available for creating non-standard (unique) device drivers? Must unique device drivers be written in Ada? [12].

8. Is there an I/O driver in the target run-time environment for communication with the host computer which can be used for target data collection (in real time)? [12].

9. Does the console monitor provide capabilities for clearing of console screens? Is there a quick response command? [12].

10. Is there support for binary I/O? [12].

11. What restrictions are there on the types that can be instantiated for I/O? [12].

12. Are there library routines for comparing one file with another? [12].

13. Are there library routines for copying from one device to another? [12].

14. What types of file organizations are possible (e.g., indexed, sequential, etc.)? [12].

15. Does the I/O implementation support parallel disk I/O, look-ahead or overlap features, or shadow recording features? [12].

16. Can one assign logical units to physical devices? [12].

17. Is there support for data base save/restore? [12].

RTS, Data Management/Capacity

1. For TEXT I/O, what are the limitations of the following: Lines/page, characters/line, and pages/file? [12].

2. What are the minimum/maximum record sizes and file sizes for the supported devices? [12].

3. Can the maximum record sizes (as established by JCL) be overridden by program specification statements? [12].

4. What buffer sizes are assigned to standard peripheral devices (line printer, disk, tape, CRT)? [12].

5. What are the maximum lengths of the various I/O queues? [12].

RTS, Data Management/Documentation

1. Is the description of the I/O system for the host and each target given? [5].

2. Does the design documentation describe the representation of the various Ada types? [5].

RTS, Data Management/Efficiency

1. Must disk records be moved from a buffer area prior to data manipulation? [12].

2. For low-level I/O, must data be moved from buffer areas prior to manipulation? [12].

3. Following completion of an I/O request, is control always returned to the requestor? [12].

4. Can another task be requested to receive control following completion of an I/O request? [12].

5. How are priorities used for I/O requests? [12].

6. How is the association of peripheral devices with files established (e.g., can there be more than one disk unit)? [12].

7. What meaning is associated with the term "external file?" [12].

COMPILER QUESTIONS

8. Are file contents always stored contiguously? [12].

9. Does the implementation recognize situations wherein a requested record already resides in memory and need not be retrieved from disk? [12].

10. Is directory information retained in main memory, or is it stored on disk until needed? [12].

11. What are the effects of using PRAGMA PACK in conjunction with other representation specifications? [12].

12. What happens to files (open/close) after execution completion? [12].

13. In record representation clauses, what restrictions are placed on the use of alignments, component clauses, etc? [12].

14. How are array types accessed? [5].

15. How are array types mapped? [5].

16. How are access types mapped? [5].

17. How are scalar types and subtypes mapped? [5].

18. How are non-discriminated records mapped? [5].

19. What effect on mapping will discriminants have on arrays? [12].

20. What sort of disk access times can be reasonably expected? [12].

21. What sort of disk access algorithms are applied (e.g., overlap seek, rhythmic seek from outside track towards center and back so as to minimize vibration seek, rhythmic seek or interleaved use of disk tracks)? [12].

22. How extendable are file structures? [12].

23. How are I/O buffers flushed automatically? [12].

RTS, Data Management/Hardware

1. For an interrupt entry, what interpretation is given to the value of an address specification? [12].

RTS, Data Management/Interfaces

1. Are I/O interfaces to a KAPSE or to the host operating system?

2. Are all interfaces for I/O functions documented?

RTS, Data Management/Rehostability

COMPILER QUESTIONS

1. Which data management modules need to be modified for a rehost?

RTS, Data Management/Retargetability

1. Which data management modules need to be modified for a retarget?

RTS, Data Management/Robustness

1. What are the effects of plugging/unplugging peripherals? [12].

RTS, Data Management/Test Availability

1. Are internal tests available to verify I/O functions and other data management implementations?

RTS, Data Management/Usability

1. Can a user request that control be returned to the calling module following an I/O request? [12].

2. Can a user cancel I/O requests? [12].

3. What are the naming conventions for files? [12].

4. Is there support for data security features (e.g., user checking, procedure validation, data sensitive checking, etc.)? [12].

5. Can a user modify the file directory on line? [12].

6. Can a user redirect I/O from a user-specified file? [12].

7. Are there any restrictions on the types that can be instantiated for I/O? [5].

RTS, Mathematical Functions/Availability

1. What user library functions (e.g., Mathematical packages are supported)? The following are taken from [13] and are provided as a guide for possible functions that may be needed:
   - Linear algebraic equations, matrix inversion.
   - Eigenvalues and eigenvectors of matrices.
   - Curve-fitting and data smoothing.
   - Statistics.
   - Function approximation methods.
   - Function minimizations.
   - Solving single non-linear equations.
   - Solving systems of linear and non-linear equations.
   - Interpolation.
   - Numerical derivatives.
   - Numerical quadrature.
   - Ordinary differential equations.
   - Partial differential equations.
   - Higher functions.

- Kalman filter.
- Digital signal processing.
- Transfer function analysis.
- Navigation functions.
- Trigonometric functions.

RTS, Mathematical Functions/Documentation

1. Are the algorithms (with references) which are used for each function documented?

2. Is the author of each of the packages given?

3. Does the documentation give the accuracy and efficiency and upper or lower limits of allowed inputs and results?

4. Does the documentation describe the handling of error situations (e.g., incorrect user input parameters, etc.)?

RTS, Mathematical Functions/Efficiency

1. Have the supplied packages been verified for correctness and efficiency? Are the efficiency results available?

2. What are the run times of the packages?

RTS, Mathematical Functions/Extendability

1. Are the packages written in Ada?

2. If written in assembler language, are there plans to translate them to Ada?

RTS, Mathematical Functions/Interfaces

1. Are package interfaces clearly documented?
   - Required input parameters?
   - Interfaces with other modules?
   - Any machine dependencies?

RTS, Mathematical Functions/Proprietary

1. Are there any proprietary or data rights restrictions on the use or distribution of any of the supplied packages?

RTS, Mathematical Functions/Rehostability

1. Are there any machine dependencies such as bit, byte, word, etc., operations? Are these documented in both the code and specifications?

RTS, Mathematical Functions/Resources Required

1. Is the size of each module given?

2. If the module requires secondary storage for its functions, are the requirements given?

COMPILER QUESTIONS

RTS, Mathematical Functions/Test Availability

1. Are tests available to verify the accuracy and efficiency of each module? Are instructions available which show the use and interpretation of the results of the tests?

RTS, Mathematical Functions/Usability

1. Does the users' documentation explain the use of each package and function?

   - Are examples given?

   - Are calling parameters (with explanations) given?

2. Is there any on-line assistance on the use of the package?

APPENDIX D

CONFIGURATION MANAGEMENT QUESTIONS

Configuration Management Requirements

Configuration Manager/Availability

1. Are there configuration management tools?

Configuration Manager/Capacity

1. Are there any limits on the number of versions or variations of an object?
2. On the total number of objects under configuration management at once?

Configuration Manager/Configuration Management

1. Are the tools configuration managed?

Configuration Manager/Costs

1. What are the costs for leasing?
2. What are the costs for purchasing?
3. Do the tools depend on any other tools having separate costs?

Configuration Manager/Documentation

1. Is the documentation complete?
2. Is it on-line?

Configuration Manager/Extendability

1. What features of the design and implementation make the tools extendable?
2. Are the tools applicable to any product of the software development process, e.g., documentation, code, etc.?

Configuration Manager/Interfaces

CONFIGURATION MANAGEMENT QUESTIONS

1. Do all interfaces with other tools utilize the database?

2. Is the interface with the compiler or any other tool mandatory?

Configuration Manager/Intraoperability

1. Are the configuration management tools integrated?

Configuration Manager/Maintainability

1. Are the design characteristics and documentation sufficient for maintenance?

Configuration Manager/Efficiency

1. What is the time required to perform typical (TBD) configuration management tasks?

2. What are the main memory and disk requirements?

Configuration Manager/Rehostability

1. What kernel interface do the CM tools assume?

2. Have the CM tools been rehosted?

Configuration Manager/Usability

1. Is the user prevented from making typical serious configuration management errors?

2. Are the error messages understandable?

3. Are the commands and outputs consistent in terminology and format?

4. Is there on-line help?

5. Is the tool an integral part of the environment, or must the user invoke special tools to achieve configuration control? [11]

6. What error messages are generated?

Configuration Manager/Proprietary

1. Is distribution of the CM tools restricted?

2. Can the government obtain and modify the source code?

Configuration Manager/Granularity

1. Are the tools composable into more powerful tools?

Configuration Manager/Test Availability

1. Are there test scenarios for typical configuration management tasks?

CONFIGURATION MANAGEMENT QUESTIONS

Configuration Manager/Maturity

1. On what production system development have these tools been used?
2. For how long? When?
3. Have the significant recommendations based on that use been implemented?
4. Are any of these applications similar in scope and requirements to the intended use?

Identification/Availability

1. Is there a distinct configuration identification tool?

Identification/Configuration Management

1. Is the tool configuration managed?

Identification/Cost

1. What are the costs for leasing? purchase?
2. Does the tool depend on any other tools having separate costs?

Identification/Documentation

1. Is the documentation complete? Is it on-line?

Identification/Extendability

1. What features of the design and implementation make the tool extendable?
2. Is the tool applicable to any product of the software development process?

Identification/Interfaces

1. Are all interfaces with other tools through the database?

Attribute Management/Availability

1. Can all objects and directories have attributes? [8]
2. Can they be set, read, and used as retrieval criteria?

Attribute Management/Capacity

1. Is there any limit on the number of attributes for an object? If so, what?

Attribute Management/Extendability

1. Can a user define a new attribute? [8]

2. Can a user define automatic attribute maintenance?

Attribute Management/Interoperability

1. Are the attributes common ones?

Attribute Management/Power

1. Are attributes inherited?

Version Management/Availability

1. Is an automatic version capability present? [8]

Version Management/Efficiency

1. Are revisions stored in toto or as a list of changes from a base? [11]

Version Management/Power

1. Can the user over-write a version assignment? [8]

2. Is a current version supported? [8]

3. Are earlier versions available?

Variation Management/Availability

1. Is a variation capability present?

Variation Management/Power

1. Is it possible to apply a single variation specification to all objects in an operation? [8]

2. Can the user specify a default variation? [8]

Relationship Management/Availability

1. Does the tool support relationships among database objects?

Relationship Management/Capacity

1. Is there any limit on the number of relationships that are supported?

Relationship Management/Completeness

1. Can objects be retrieved on the basis of their relationships?

2. Is a relationship maintained in spite of name changes? [8]

Relationship Management/Extendability

1. Can the user define new relationships among objects?

Configuration Control/Availability

CONFIGURATION MANAGEMENT QUESTIONS

1. Is there a distinct configuration control tool?

Configuration Control/Configuration Management

1. Is the tool configuration managed?

Configuration Control/Costs

1. What are the costs for leasing? purchase?
2. Does the tool depend on any other tools having separate costs?

Configuration Control/Documentation

1. Is the documentation complete?
2. Is it on-line?

Configuration Control/Extendability

1. What features of the design and implementation make the tool extendable?
2. Is the tool applicable to any product of the software development process?

Configuration Control/Interfaces

1. Are all interfaces with other tools through the database?

Workspace Partitioning/Availability

1. Can the database be partitioned into multiple distinct projects and protected from interference by unique naming and access controls?

Workspace Partitioning/Interfaces

1. Is there a consistent interface between an individual developer and the rest of the project? [8]

Access Control/Availability

1. What types of access rights are provided?

Access Control/Extendability

1. Can the types of access rights be extended? [8]

Access Control/Power

1. Can access be controlled at the directory as well as object level? [8]

Access Control/Completeness

1. Can access be controlled on project, subproject, and individual basis? [8]

CONFIGURATION MANAGEMENT QUESTIONS

2. Can the user specify default access rights by team or individual? [8]

3. Does the user have the ability to change access rights, or to control who can change access rights? [8]

Baseline Management/Availability

1. Is there a mechanism for defining a baseline?

2. For controlling modifications to the baseline?

Baseline Management/Power

1. Can all components of a baseline be identified by a single reference? [8]

Protection/Availability

1. Is there a configuration management database backup capability?

Protection/Efficiency

1. Are partial and incremental backups supported?

Protection/Robustness

1. Is deletion of an object prohibited as long as a derived object exists?

Status Accounting and Reporting/Availability

1. Can the content and status of a baseline be determined directly?

History Reporting/Availability

1. Can the history of an object be obtained directly?

History Reporting/Extendability

1. Can the user define the content of the history report?

Configuration Reporting/Availability

1. Is there a mechanism for reporting on the content and status of a configuration baseline?

APPENDIX E

E&V TEAM REQUIREMENTS WORKING GROUP MEMBERSHIP

Co-Chairpersons:	Betsy Bailey	Institute for Defense Analysis
	Tim Lindquist	Virginia Tech
Members:	Greg Bettice	Naval Avionics Ctr.
	Capt R. Contreras	Kirtland AFB
	Hubert Dorsett	Kelly AFB
	Rich Fleming	Aerospace Corp.
	Robert Fritz	Computer Sciences Corp.
	Marlene Hazel	Mitre Corp.
	Ronnie Martin	Georgia Tech
	John Miller	McClellan AFB
	Mike Meirink	Sperry Corp.
	Amos Rohrer	EG_&G
	Helen Romanowsky	Rockwell International
	Ray Sandborgh	Sperry Corp.

APPENDIX C

DOD APSE ANALYSIS DOCUMENT

Version 1.0
28 September 1984

Prepared by

Evaluation & Validation Team

APSE Working Group

for the

Ada Joint Program Office

TABLE OF CONTENTS

1.0 EXECUTIVE SUMMARY

1.1 The overall Evaluation and Validation (E&V) Task objective is to develop the technology for the evaluation and validation of APSEs. In support of the E&V task, the APSE Working Group (APSEWG) (see Appendix E for a list of APSEWG members) was formed to obtain expertise on DoD developed APSEs, identify capabilities/tests/tools associated with currently available software programming environments, and compare the capabilities with the DoD APSEs. A DoD APSE is an Ada Programming Support Environment that is/was developed using Government funding and according to Government specifications. The DoD is currently developing two Minimal Ada Programming Support Environments (MAPSEs) the Ada Integrated Environment (AIE) and the Ada Language System (ALS). A MAPSE is an APSE with the minimal features implemented to support software development and maintenance. As a result, this document was produced to identify existing programming support environments for the Ada language and provide a taxonomy of the capabilities of each environment. The environments identified and described herein include the Air Force's Ada Integrated Environment (AIE), the Army's Ada Language System (ALS), and the Navy's Ada Language System/Navy (ALS/N). It is anticipated that future versions of this document will include a comparison of the functional capabilities of the identified environments, the evaluation criteria, and an analysis of the application of the criteria to the existing DoD APSEs. At this time, the document identifies areas for future investigation and analysis.

2.0 INTRODUCTION

2.1 The Ada Joint Program Office (AJPO) is sponsoring a tri-service APSE E&V Team with the Air Force designated as the lead service. The E&V Team has been tasked to develop a capability to perform assessment of APSEs and to determine conformance to the Common APSE Interface Set (CAIS) being developed by the KAPSE Interface Team/KAPSE Interface Team Industry and Academia (KIT/KITIA). In order to accomplish the E&V goal, several subtasks have been identified. Some include the development of requirements for APSE E&V and the development of an APSE E&V classification schema in which APSE components, interface areas, and appropriate evaluation or validation criteria associated with each APSE component will be identified. APSE components will be identified and classified based upon the existence of criteria, standards and metrics capabilities for each component. An APSE evaluation capability, APSE validation capability, and procedures for the implementation of E&V will be developed. The E&V team will monitor the formal qualification testing of DoD APSEs and develop evaluation and validation tools and aids.

2.2 As described in the STONEMAN document [1], the purpose of an APSE is to support the development and maintenance of Ada applications software throughout its life cycle with particular emphasis on software for mission critical applications. STONEMAN notes that an Ada programming support environment that provides the minimal functional components necessary for the generation, compilation and execution of Ada programs is called a minimal APSE or MAPSE. To date, the DoD has undertaken the development of MAPSEs and this document identifies and describes these MAPSEs. The features of the MAPSEs are categorized according to the outline provided in "A Taxonomy of Tool Features for the Ada Programming Support Environment (APSE)" [2].

2.3 Future versions of this document will include a comparative analysis of the identified APSEs (section 6.0), evaluation criteria for DoD APSEs developed with respect to these APSEs, and an analysis of the application of the criteria to the APSEs (section 7.0). In this version, these sections will include the areas and relevant issues related to E&V which will guide this working group in the comparison of the identified APSEs and development of evaluation criteria for the components of the identified APSEs. They do not include the only areas and issues to be addressed, but rather a starting point for analysis. The E&V Requirements Document, developed by the E&V Requirements Working Group (REQWG) will be used for further determination of areas to be investigated.

3.0 SCOPE

3.1 The purpose of the DoD APSE Analysis Document is to provide descriptions and taxonomies of the features provided in Ada programming support environments (APSEs) developed by the DoD. Future versions of this document will address a combined taxonomy of the APSES, evaluation criteria (in particular performance measurement criteria), and a comparative analysis of the identified APSEs. Should a new DoD developed APSE become available to this working group, its description, taxonomy, and comparative analysis will be included in subsequent versions of this document.

4.0 IDENTIFICATION OF APSEs

4.1 In order to evaluate and validate an APSE, the first question that arises is "What constitutes an APSE?". STONEMAN defines an APSE as an integrated programming environment. The three principal features include the data base, interfaces, and the toolset. According to STONEMAN, the minimal toolset or MAPSE includes a text editor, prettyprinter, translator(s), linker(s), loader(s), set-use static analyser, control flow static analyser, dynamic analysis tool, terminal interface routines, file administrator, command interpreter, and configuration manager. An analysis of existing programming support systems (both DoD developed and commercially available) that support the generation, compilation, and execution of Ada source programs is on-going to determine the functional capabilities available (or potentially available) that will require evaluation. The DoD Ada Programming Support Environments that are under-going analysis include the Ada Integrated Environment (IBM 4341/VM) the Ada Language System (VAX 11/780/VMS), and the Ada Language System/Navy (VAX/VMS). The following section provides a brief overview of the functional components in each of the systems available. A detailed description may be found in Appendix A.

5.0 APSE OVERVIEW

5.1 The Ada Integrated Environment (AIE).

In April 1982, the Air Force (Rome Air Development Center) contracted Intermetrics, Inc. to implement a Minimal Ada Programming Support Environment (MAPSE) entitled "Ada Integrated Environment" (AIE). The AIE is designed for use in the development of embedded computer system software and can accommodate a variety of users, skilled and unskilled, from project managers, program designers and developers to documentors and clerical personnel. The AIE contains a virtual operating system called the kernel or KAPSE (Kernel Ada Programming Support Environment) that isolates tools (both system and user) from hardware dependencies. The system tools consist of a production quality Ada Compiler and symbolic debugger targetted to the IBM 4341; a program integration facility with program library management and linking/loading tools used to develop Ada programs; a data base manager with a complete file management system; and a command language processor which allows user interaction with tools and other operating system routines. The AIE will be hosted on the IBM 4341 and can co-exist with other operating systems (e.g., OS/VS1, CMS, UTS, etc.).

5.2 The Ada Language System (ALS)

In April, 1980, the Army contracted SofTech, Inc. to develop the Ada Language System (ALS), an integrated programming environment designed to aid in the development and maintenance of Ada programs. The ALS is designed to support large software systems throughout their life cycle. In particular, the ALS was designed with the requirements of embedded computer system development in mind. The three major components include a file structure (called the environment database), a set of tools, and a mechanism through which the tools are invoked (i.e., the command language interpreter).

5.3 Ada Language System/Navy (ALS/N)

The Ada Language System/Navy is a minimal Ada programming support environment designed to provide support for program generation and execution of Ada application programs targetted for Navy standard embedded computers and peripherals. The environment described in Appendix A represents the first phase implementation of a full Ada Programming Support Environment (APSE) for the Navy. The system is composed of extensions to the Army' ALS that are the minimum required to support projects using the Navy's standard embedded computers.

6.0 COMPARATIVE ANALYSIS OF APSEs

6.1 The APSEs identified and described above will be compared to determining evalution criteria that may be incorporated in the E&V approach. This section will include an analysis of the results of the comparison. For example, in the development of this document, it became apparent that the documentation for each of the environments was radically different with respect to the level of detail in the B-5 specifications, users manuals, etc. that was available. The documentation for each environment will be analyzed more closely to determine evaluation criteria specific to documentation practices, procedures, etc. The issue of what constitutes "good" documentation will be addressed. The issues of objectivity and subjectivity will be investigated as potential hazards in determine the evaluation criteria.

6.2 It is NOT the intention of this working group to proclaim any one of the above-mentioned environments better or worse than any of the others. Each of the environments is in a different stage of development and was developed for a different purpose. Since at this time, the AIE and ALS/N are in the design stage, it is impossible to determine the size, speed, or implementability of these environments. The ALS was released as a preliminary version for rehosting and retargetting purposes and is by no means complete. Comparisons between these environments are meaningless at this time. However, this group intends to become familiar with the above environments in order to determine their functional capabilities and to develop potential evaluation criteria.

7.0 EVALUATION CRITERIA AND ANALYSIS

7.1 At a first glance, it can be seen that each of the above environments has a similar set of functional characteristics and yet the approach taken to implement these functions is different. As well, there are features in one environment that are missing in the others. It is the intention of this working group to develop a combined taxonomy which will incorporate all the existing features. As well, software environments that do not support the compilation of Ada source programs will be investigated for functional capabilities that may eventually be incorporated in an APSE. This section will describe the evaluation criteria identified in the above-mentioned studies.

APPENDIX A

A.1 ADA INTEGRATED ENVIRONMENT (AIE) DESCRIPTION

A.1.1 The overall structure of the AIE database is hierarchically relational and provides many primitives and operations for use in the development of project configurations. The database contains a collection of objects that have attributes and content. The attributes of an object distinguish it from other objects in the database. There are three classes of objects: simple, composite and window. The content of a simple object is an Ada external file. The content of a composite object is a collection of component objects which may be simple, composite or window objects. The content of a window object is a cross reference to a partition of another object in the database. It is the mechanism through which access to and responsibility for specific parts of the database is permitted.

A.1.1.1 The attributes of an object are the most important means of partitioning and building the database. They describe the purpose, content, and access of an object. In the AIE database, there are system and user defined attributes. The system defined attributes are category, access control and history as defined in STONEMAN. There are two kinds of user defined attributes, distinguishing and non-distinguishing. The user defined distinguishing attributes are the mechanism for the distinction between objects in the database. They are later used to select various components from the composite object.

A.1.2 The MAPSE Ada compiler processes the full Ada language and is designed to be rehostable and retargetable. It is divided into three logically separate phases, the front end, the middle part, and the back end. The front end is separated into two processing phases that may be invoked separately. The first phase (LEXSYN) performs lexical and syntactic analysis and generates an abstract syntax tree and name table. To assist in the development of LEXSYN, two tools are used, the lexer and parser generators which accept as input formal grammars and output tables and skeleton recognizer programs that work from these tables. The second phase performs semantic analysis, completes the symbol table and generates an intermediate representation called DIANA. One of the major differences of the Intermetrics DIANA from the Tartan Laboratories DIANA [18] is that the class structure has been changed so that classes form a strict hierarchy and lists of nodes have been removed since they contain no additional information. The DIANA for each Ada compilation unit is stored in the program library when the front end accepts the source code as valid Ada text. The middle part of the compiler implements generic instantiations and determines if code sharing among instances is possible. It also adds static information for the LIST and OPTIMIZE compiler options and lowers the semantic level of the DIANA tree, making it more machine-oriented, into the BILL tree. The back end phase transforms a BILL tree into an equivalent linkable object module and performs target machine independent optimizations such as constant propagation,

redundant constraint check elimination, dead code elimination, strength reduction, etc. as well as machine dependent optimizations like peephole optimization and branch resolution (using shorter relative branch instructions instead of full word branch instructions).

A.1.3 Independent and modular program development is supported by the Program Integration Facility, (PIF). PIF occurs in two phases, compilation and program building. First, the Ada compiler processes a single compilation unit in the context of other compilation units that have been processed before. The compiler must access the program library in order to obtain information about the compilation units used by the current unit being processed. Upon successful compilation, the compiler updates the program library with the information gathered while compiling. The PIF provides functions to allow the compiler to access the program library during the processing of single compilation units. The second phase, performed by the Program Builder, generates a complete executable program from the separately compiled units placed in the program library. The Preamble Generator is invoked to automatically generate an Ada parameterless procedure whenever the main program specified by the user was a function or a procedure with parameters. The Program Completeness Checker verifies that all compilation units used are available in object form and are consistent. The Body Generator may be invoked to create a null subprogram, package or task body if the body of a corresponding specification does not exist in the program library. The Body Generator may be invoked directly by the user to create a body skeleton. Finally, the Linker is invoked to create the executable program by linking the main program and all referenced object modules, resolving all external references. In addition, the PIF provides other tools for the manipulation of program libraries. The user may create, copy, display information about, or delete program libraries. As well, the user may create, copy, examine, and delete the contents of a program library.

A.1.4 The MAPSE Command Processor provides the interface between the user and the MAPSE tools, giving the user a means of executing the tools. It is an Ada program that interprets commands, written in MCL (MAPSE Command Language), on a line by line basis. Commands may be executed in the foreground or background; entered interactively or stored as scripts for later execution; and may be interrupted and restarted by the user. The MCL uses Ada-like constructs whenever possible and allows for user defined variables as well as predefined variables. Predefined status variables include EXECUTION_TIME (the execution time of the last foreground command), EXIT_STATUS (the exit status of the last completed foreground job command), ACTIVE_TASKS (a list of all currently executing tasks), etc. The complete list is included in the AIE taxonomy (see Appendix A).

A.1.5 The MAPSE Debugger (DBUG) is an Ada program that includes a command processor that accepts as input commands from the Debug Command Language, breakpoint command procedures, execution control procedures, information command procedures, utility procedures, and program library access procedures. The breakpoint command procedures

provide the necessary facilities to create and maintain user defined breakpoints. The functions provided within execution control procedures include stopping at a breakpoint, executing any commands (or command scripts) associated with the breakpoint, controlling a STEP function, and proceeding with the correct user program address. The information command procedures provide the facilities for displaying variables, program text, etc. The utility procedures include functions and procedures for storing and communicating with other programs (e.g., program integration tools). Access to the symbol and statement tables, cross reference and linking information, and source listings are provided through the program library access procedures.

A.1.6 The MAPSE Text Editor is invoked as a standard Ada program. The editor may be utilized in various modes depending upon the terminal device and experience of the user. The command mode provides the basic editing capabilities for the novice user, whereas the screen mode is a superset of the command mode and provides CRT terminal capabilities. Upon invoking the editor, the user is automatically put into command mode and may invoke the more sophisticated features as necessary during the edit session. Additional capabilities provided include the ability to escape to the KAPSE (Operating System) level and pass portions of the edited text as input to a user program. As well, the results of the program execution may be read into the edit buffer (temporary workspace) in which editing can be resumed.

A.1.7 The AIE provides most of the toolset called out in the STONEMAN document and is designed to be open-ended to enable the addition of new tools. It is constructed and maintained with the use of the MAPSE Generation and Support tool. This tool contains the bootstrap facility, the parser and lexer generators, and the facilities required to rehost the AIE. The Virtual Memory Methodology tool allows for the creation and manipulation of abstract data structures without machine dependencies.

A.1.8 AIE TAXONOMY

1.0 MANAGEMENT
  1.1 Configuration Control
    1.1.1 Attribute Functions
      1.1.1.1 set an attribute
      1.1.1.2 get an attribute
      1.1.1.3 get all attributes
    1.1.2 Window Functions
      1.1.2.1 create a window
      1.1.2.2 delete a window
      1.1.2.3 copy a window
      1.1.2.4 rename a window
      1.1.2.5 get window id
      1.1.2.6 get next child window
      1.1.2.7 get info on window
      1.1.2.8 revoke a window
    1.1.3 Partition Functions
      1.1.3.1 open partition
      1.1.3.2 close partition
      1.1.3.3 get next component in partition
      1.1.3.4 list the partition
    1.1.4 Role Functions (Access Control)
      1.1.4.1 create a role
      1.1.4.2 delete a role
      1.1.4.3 set role access
      1.1.4.4 get role access
      1.1.4.5 get all roles
      1.1.4.6 adopt a role
      1.1.4.7 abandon a role
      1.1.4.8 give a role
    1.1.5 History and Archiving Functions
      1.1.5.1 get history reference (current "state" of object)
      1.1.5.2 create new source archive
      1.1.5.3 recreate the contents and user attributes of archived object
      1.1.5.4 assign an object to existing source archive
      1.1.5.5 get states from which the specified state is directly derived
      1.1.5.6 get source states from which the specified state is directly or indirectly derived
      1.1.5.7 get the history parameters
      1.1.5.8 bring history script or archive on-line
      1.1.5.9 check if history script or archive is already on-line
      1.1.5.10 get history time/date
      1.1.5.11 get user name associated with specified script or source state
      1.1.5.12 get (name of) most recent revision
    1.1.6 Access synchronization
      1.1.6.1 reserve an object
      1.1.6.2 release an object
      1.1.6.3 abort the reservation of an object

1.1.6.4 determine the user who performed reservation

1.2 Information Management

1.2.1 Ada Library Management (Program Library Tools)

1.2.1.1 Program Library Manager

1.2.1.1.1 Collection Functions

1.2.1.1.1.1 create a collection
1.2.1.1.1.2 delete a collection
1.2.1.1.1.3 link collections
1.2.1.1.1.4 unlink collections
1.2.1.1.1.5 add rules for collections
1.2.1.1.1.6 delete rules for collections
1.2.1.1.1.7 modify rules for collections
1.2.1.1.1.8 add approved operations of a collection
1.2.1.1.1.9 delete approved operations of a collection
1.2.1.1.1.10 modify approved operations of a collection
1.2.1.1.1.11 display segment number information of a collection
1.2.1.1.1.12 display objects within collection
1.2.1.1.1.13 display resource catalog information (name, prefix set, version/revision, etc)

1.2.1.1.2 Catalog Functions

1.2.1.1.2.1 create a catalog
1.2.1.1.2.2 delete a catalog
1.2.1.1.2.3 promote a catalog
1.2.1.1.2.4 derive a catalog
1.2.1.1.2.5 copy a catalog
1.2.1.1.2.6 create unit-name prefix set for a catalog
1.2.1.1.2.7 update unit-name prefix set for a catalog
1.2.1.1.2.8 link to a resource catalog
1.2.1.1.2.9 unlink from a resource catalog
1.2.1.1.2.10 update catalog links
1.2.1.1.2.11 display type of the catalog (primary or resource, interface/implementation)
1.2.1.1.2.12 display the prefix set of a catalog
1.2.1.1.2.13 display resource catalogs referenced within the catalog
1.2.1.1.2.14 display all objects within the catalog

1.2.1.1.3 Library Unit Functions

1.2.1.1.3.1 delete a library unit
1.2.1.1.3.2 save a library unit
1.2.1.1.3.3 bring a library unit up-to-date
1.2.1.1.3.4 display library unit's precursor list
1.2.1.1.3.5 display library unit's requirements list
1.2.1.1.3.6 display whether the library unit is up-to-date
1.2.1.1.3.7 display the date the library unit was created
1.2.1.1.3.8 display the size of the library unit

1.2.1.2 Link map/X-ref Lister

1.2.1.3 Unit Lister

1.2.1.4 Help Installer

1.2.2 Specification Management

no provision
1.2.3 Data Dictionary Management
no provision
1.2.4 Ada Package Management
no provision
1.2.5 Test Management
no provision
1.3 Project Management
1.3.1 Cost Estimation
1.3.2 Scheduling
1.3.3 Tracking
1.4 System Management
1.4.1 Backup and Recovery
1.4.1.1 Full Backup
1.4.1.2 Incremental Backup
1.4.1.3 Recovery of objects
1.4.2 Terminal I/O
1.4.2.1 read from specified terminal
1.4.2.2 write to specified terminal
1.4.2.3 set terminal characteristics
1.4.2.4 get terminal characteristics
1.4.3 Device I/O
1.4.3.1 open device
1.4.3.2 read device
1.4.3.3 write device
1.4.3.4 close device
1.4.3.5 get information on device
1.4.3.6 set information about device
1.4.4 Interactive I/O
1.4.4.1 set cursor & echoing of input at current line & col of output
1.4.4.2 break any echoing association
1.4.4.3 get output information (e.g., terminal's height & width)
1.4.4.4 set output information
1.4.4.5 get input information (e.g., keyboard control characters)
1.4.4.6 set input information
1.4.5 Multiple Program Management
1.4.5.1 Program Loading
1.4.5.1.1 load program
1.4.5.1.2 unload program
1.4.5.1.3 allocate storage
1.4.5.1.4 free storage
1.4.5.2 KAPSE Program Communication
1.4.5.2.1 KAPSE Call - signal that a message should be sent across KAPSE protection boundary. KAPSE receives request and returns results.
1.4.5.3 Program Invocation
1.4.5.3.1 invoke executable program context/CL script
1.4.5.3.2 search for executable program context or CL script
1.4.5.3.3 invoke program but caller is not suspended until completion

1.4.5.3.4 wait for completion of specified program context
1.4.5.3.5 exit a program
1.4.5.3.6 suspend a program
1.4.5.3.7 resume execution of a program
1.4.5.3.8 extract a parameter from a parameter string
1.4.5.3.9 invoke a specified operation
1.4.5.4 Inter-Program Communication
1.4.5.4.1 accept next entry call for specified channel
1.4.5.4.2 end rendezvous and provide results
1.4.5.4.3 send parameters to designated context via channel
1.4.6 Virtual Memory Methodology
1.4.6.1 Rep Analyzer - operations generated:
1.4.6.1.1 operations on virtual record types
1.4.6.1.1.1 set root locator value for a VMSD
1.4.6.1.1.2 obtain root locator value for a VMSD
1.4.6.1.1.3 iterate over all virtual records within a VMSD
1.4.6.1.1.4 output a sequence of domains in virtual record notation
1.4.6.1.1.5 output a sequence of VMSDs in virtual record notation
1.4.6.1.1.6 output single virtual record in virtual record notation
1.4.6.1.1.6 read virtual record notation into one or more domains
1.4.6.1.2 operations on vectors
1.4.6.1.2.1 create a vector
1.4.6.1.2.2 access an element of a vector
1.4.6.1.2.3 set a value of a vector
1.4.6.1.2.4 obtain a value of a vector
1.4.6.1.2.5 obtain the size of a vector
1.4.6.1.2.6 delete a vector
1.4.6.1.3 operations on lists
1.4.6.1.3.1 create a list
1.4.6.1.3.2 create a cell
1.4.6.1.3.3 access a cell
1.4.6.1.3.4 obtain a value of a cell
1.4.6.1.3.5 set a value of a cell
1.4.6.1.3.6 locate the next or previous cell
1.4.6.1.3.7 insert a cell
1.4.7.1.3.8 remove a cell
1.4.6.1.3.9 append one list to another
1.4.6.1.3.10 delete a list
1.4.6.1.4 operations on sets
1.4.6.1.4.1 create a set
1.4.6.1.4.2 add members to set
1.4.6.1.4.3 find the value of a member
1.4.6.1.4.4 find specified values
1.4.6.1.4.5 copy sets
1.4.6.1.4.6 find the next member in a set
1.4.6.1.4.7 the size of a set
1.4.6.1.4.8 determine intersection of sets

1.4.6.1.4.9 determine union of sets
1.4.6.1.4.10 determine difference of sets
1.4.6.1.4.11 determine symmetric difference of sets
1.4.6.1.4.12 delete a set
1.4.6.1.5 operations on text strings
1.4.6.1.5.1 create a string
1.4.6.1.5.2 delete a string
1.4.6.1.5.3 access text value
1.4.6.2 VMM Basic Operations
1.4.6.2.1 open a domain
1.4.6.2.2 close a domain
1.4.6.2.3 create a subdomain (VMSD)
1.4.6.2.4 access a subdomain (VMSD)
1.4.6.2.5 close a subdomain (VMSD)
1.4.6.2.6 create a node
1.4.6.2.7 reference a node
1.4.6.2.8 change a node kind
1.4.6.2.9 get root node of a subdomain
1.4.6.2.10 set root node of a subdomain

2.0 STATIC ANALYSIS
2.1 Type Analysis
2.1.1 Compiler Front-end - syntax and semantic analysis
2.2 Interface Analysis
2.2.1 Compiler - specification/body matching
2.3 Statistical Profiling
no provision
2.4 Cross Reference
2.4.1 PIF Lister - cross reference listing (see above)
2.5 Auditing
no provision
2.6 Complexity Measurement
no provision
2.7 Completeness Checking
no provision
2.8 Consistency Checking
no provision
2.9 Structure Testing
no provision
2.10 Reference Analysis
no provision

3.0 DYNAMIC ANALYSIS
3.1 Timing Analysis
no provision
3.2 Tuning Analysis
no provision
3.3 Tracing/Debugging
3.3.1 Debugging Capabilities
3.3.1.1 Breakpoint commands
3.3.1.1.1 set breakpoint
3.3.1.1.1.1 break after specified statements/labels
3.3.1.1.1.2 break before specified statements or labels

3.3.1.1.1.3 break before each statement that modified the specified variables
3.3.1.1.1.4 break on raise of specified exceptions
3.3.1.1.1.5 break on raise of all exceptions
3.3.1.1.1.6 break on unhandled exceptions
3.3.1.1.1.7 break on entry to all subprograms and entries
3.3.1.1.1.8 break on exit from all subprograms and entries
3.3.1.1.2 remove breakpoints
3.3.1.1.3 suspend breakpoints
3.3.1.1.4 restore breakpoints (only those suspended)
3.3.1.2 Execution Control
3.3.1.2.1 start execution of a program
3.3.1.2.1.1 proceed from the specified label or statement
3.3.1.2.1.2 return from subprogram
3.3.1.2.1.3 continue the user program from where it was suspended
3.3.1.2.1.4 ignore breakpoint the next specified times
3.3.1.2.1.5 invoke a user program which must be in the load module
3.3.1.2.1.6 raise the specified exception and proceed
3.3.1.2.2 stop execution of a program
3.3.1.2.3 modify execution of a program
3.3.1.2.3.1 delay a task for specified time period (seconds)
3.3.1.2.3.2 abort the specified task
3.3.1.2.3.3 change the priority of a task
3.3.1.3 Information commands
3.3.1.3.1 display values of specified variables
3.3.1.3.2 display values of all variables in scope
3.3.1.3.3 display status of tasks at specified levels starting at current block with any or all of the following options:
3.3.1.3.3.1 tasks currently activated
3.3.1.3.3.2 tasks that are running
3.3.1.3.3.3 all tasks waiting at any call point (or at a call point for a particular entry)
3.3.1.3.3.4 all tasks waiting at any accept point or (for a particular entry call)
3.3.1.3.3.5 tasks that are waiting because of a program delay
3.3.1.3.3.6 tasks which are dependent on the current block
3.3.1.3.3.7 tasks which are blocked and why
3.3.1.3.4 display any or all breakpoints
3.3.1.3.5 display specified lines of source code
3.3.1.3.6 display command and script associated with specified breakpoint id
3.3.1.4 DBUG control
3.3.1.4.1 initiate DBUG processing

3.3.1.4.1.1 set trace option on (default) or off (options are chain, flow, task and all)
3.3.1.4.1.2 set default base for output of variables
3.3.1.4.1.3 verbose (on or off, default is on) prompt
3.3.1.4.2 suspend DBUG processing
3.3.1.4.3 read in command files
3.3.1.4.4 redirect output
3.3.1.4.4.1 append a copy of breakpoints and display information to a file
3.3.1.4.4.2 save current breakpoints in file
3.3.1.4.5 change scope to caller of current scope
3.3.1.4.6 change scope to static enclosing scope
3.3.1.4.7 reset scope to original breakpoint scope
3.3.1.4.8 invoke command processor
3.3.1.4.9 invoke editor on the commands associated with specified id
3.3.1.4.10 execute file as a stream of DBUG commands
3.3.1.4.11 stop user program from execution and return control to DBUG

3.4 Regression Testing
no provision

3.5 Assertion Checking
no provision

3.6 Coverage Analysis
no provision

4.0 TRANSFORMATION

4.1 Formatting

4.1.1 Source Reconstructor
4.1.1.1 reconstruct source from DIANA or AST verbatim
4.1.1.1.1 user specifies indentation (default is one tab per level)
4.1.1.1.2 user specifies commenting indentation (default is one tab to right of current indentation column)
4.1.1.2 reconstruct source from DIANA or AST according to Ada LRM conventions

4.2 Optimization

4.2.1 Optimization Level
4.2.1.1 Time - perform all optimizations
4.2.1.2 Space - perform all passes, but eliminate strength reduction
4.2.1.3 None - perform no optimizations

4.2.2 Optimizations Performed
4.2.2.1 constant propagation
4.2.2.2 redundant constraint check elimination
4.2.2.3 constant folding
4.2.2.4 elimination of unreachable code
4.2.2.5 code motion - movement of loop invariant code out of loops
4.2.2.6 redundant computation elimination
4.2.2.7 strength reduction - reducing multiplications within loops to additions
4.2.2.8 algebraic simplifications of expressions and

statements
4.2.2.9 peephole optimization
4.3 Compilation
4.3.1 Compiler Options
4.3.1.1 LIST =>
4.3.1.1.1 enable/disable a listing (default is disable)
4.3.1.1.2 generate/suppress text listing (default is list)
4.3.1.1.3 generate/suppress listing of symbol table attributes of identifiers (default is suppress listing)
4.3.1.1.4 generate/suppress cross reference listing of all identifiers (default is suppress listing)
4.3.1.1.5 generate/suppress assembly code listing (default is suppress listing)
4.3.1.2 LISTERRS => n
print errors above severity n in listing (default is 0)
4.3.1.3 TTYERS => n
print errors above severity n on terminal (default is 0)
4.3.1.4 NOSEM => n
if syntax errors >= n occur, suppress the rest of the phases (default is 50)
4.3.1.5 NOCODE => n
if semantic errors >= n occur, suppress the rest of the phases (default is 50)
4.3.1.6 DEBUG =>
4.3.1.6.1 allow/disallow DBUG to alter and inspect information
4.3.1.6.2 insert/suppress insertion of DBUG hooks after each statement and at the beginning and end of each procedure
4.3.1.7 OPTIMIZE =>
4.3.1.7.1 for space
4.3.1.7.2 for time
4.3.1.7.3 no optimizations are performed
4.3.1.8 STATISTICS =>
collect statistics (types are TBD)
4.3.1.9 COMMENT =>
preserve/remove comments in the DIANA (default is to remove)
4.3.1.10 REORDER =>
allow/disallow the compiler to reorder compilation units (default is to disallow)
4.3.1.11 SPACE =>n
allow compiler n kilobytes space (default: 512)
4.3.1.12 LOOKAHEAD => n
n is the number of tokens to lookahead in parsing (default is 5)
4.3.1.13 TRACE =>
used by compiler developers and maintenance only
4.3.2 Functional Components

4.3.2.1 Driver - call Front, Middle and Back End in order or separately
  4.3.2.2 Front End
    4.3.2.2.1 Lexsyn Phase - performs lexical and syntactic analysis
    4.3.2.2.2 Sem Phase - performs semantic analysis
  4.3.2.3 Middle Part
    4.3.2.3.1 Geninst Phase - performs generic instantiation
    4.3.2.3.2 Statinfo Phase - constructs the call graph and symbol cross references
      4.3.2.3.3 Storage Phase - determines the run-time representation for data and the principal storage requirements
    4.3.2.3.4 Expand Phase - carries out a major tree rewrite that removes the implicit Ada semantics and exposes address arithmetic
  4.3.2.4 Back End
      4.3.2.4.1 Flow Phase - performs machine independent optimizations
      4.3.2.4.2 Vcode Phase - performs a tree walk simulating code generation and determines the register requirement
      4.3.2.4.3 Tnbind Phase - determines location of every object the code generator will deal with
      4.3.2.4.4 Codegen Phase - uses machine-specific templates to generate a linked list of locally optimal target machine instructions
      4.3.2.4.5 Final phase - performs machine dependent "peephole" optimizations, branch resolution and cross jumping

4.3.3 Program Builder
  4.3.3.1 Builder
      4.3.3.1.1 parameters:
        4.3.1.1.1 lib_update - specify whether units that are not up-to-date should be brought up to date (default is on)
        4.3.1.1.2 unit_gen - specify whether missing bodies should be generated and compiled (default is on)
        4.3.1.1.3 csect_elim - specify whether the linker should eliminate unreferenced csects from its output (default is on)
        4.3.1.1.4 xref - specify whether xref information should be generated for the link map/xref lister (default is off)
      4.3.3.1.2 check program completeness and consistency (call program completeness checker)
      4.3.3.1.3 traditional linking function
  4.3.3.2 Preamble Generator
    4.3.3.2.1 determine the need for a preamble
    4.3.3.2.2 generate the preample (if necessary)
  4.3.3.3 Program Completeness Checker
    4.3.3.3.1 verifies completeness
      4.3.3.3.2 creates a minimal body for unit if necessary

4.3.3.3.3 update object module if necessary
4.3.3.4 Body Generator
4.4 Editing
4.4.1 Syntax Directed
no provision
4.4.2 Basic Editor
4.4.2.1 line command mode functions
4.4.2.1.1 display functions
4.4.2.1.1.1 display current line
4.4.2.1.1.2 display current line number
4.4.2.1.1.3 display specified line(s)
4.4.2.1.1.4 display specified line(s) with line numbers
4.4.2.1.1.5 display line(s) with + or - offset
4.4.2.1.1.6 display current file name and number of lines in file
4.4.2.1.2 mark lines
4.4.2.1.3 text manipulation
4.4.2.1.3.1 copy specified lines
4.4.2.1.3.2 replace specified lines with text input
4.4.2.1.3.3 substitute specified string with specified string
4.4.2.1.3.4 delete specified lines
4.4.2.1.3.5 globally search for patterns
4.4.2.1.3.6 insert/append text
4.4.2.1.3.7 join text from specified range of lines to one line
4.4.2.1.3.8 move text to another part of buffer
4.4.2.1.3.9 delete text and place in temporary buffer
4.4.2.1.3.10 insert text from temporary buffer into specified lines
4.4.2.1.4 read from a file
4.4.2.1.5 write to a file (all or part of edit buffer)
4.4.2.1.6 write specified lines as input to specified MCP command
4.4.2.1.7 undo the changes made from the last editing command
4.4.2.1.8 quit
4.4.2.2 full screen mode
4.4.2.2.1 all line oriented commands
4.4.2.2.3 scroll up/scroll down
4.4.2.2.4 move forward/backward a page
4.4.2.2.5 searching functions
4.4.2.2.5.2 search for specified string and position cursor at first occurrence moving forward or backward in buffer
4.4.2.2.5.3 go to next line with an occurrence of a previously mentioned string
4.4.2.2.5.4 go to beginning of file
4.4.2.2.5.5 go to end of file
4.4.2.2.5.6 go to specified line
4.4.2.2.6 moving around the screen
4.4.2.2.6.1 advance to next line

4.4.2.2.6.2 advance to previous line
4.4.2.2.6.3 move to top line (home)
4.4.2.2.6.4 move to middle of the screen
4.4.2.2.6.5 move to last line of screen
4.4.2.2.6.6 move to next line maintaining column position
4.4.2.2.6.6 move to previous line maintaining column position
4.4.2.2.6.7 move to next word
4.4.2.2.6.8 move back one word
4.4.2.2.6.9 advance to end of current word
4.4.2.2.6.10 move right one character
4.4.2.2.6.11 move left one character
4.4.2.2.6.12 move forward/backward one sentence
4.4.2.2.6.13 move forward/backward one paragraph
4.4.2.2.3 open new lines before/after the current line
4.4.2.2.4 delete character under the cursor
4.4.2.2.5 replace character under cursor position with next character typed
4.4.2.2.6 substitute characters for the character under the cursor
4.4.2.2.7 clear screen
4.4.2.2.8 erase to end of line
4.4.2.2.9 hardware character and line insert/delete
4.4.2.3 KAPSE or user program invocation
4.4.2.3.1 escape to KAPSE level - or user level
4.4.2.3.2 pass portions of edited object to program as input
4.4.2.3.3 output of invoked program may be read into edit buffer
4.4.2.4 user controlled options
4.4.2.4.1 specify length of tabstops
4.4.2.4.2 specify automatic wraparound of text to next line (default)
4.4.2.4.3 specify automatic indentation (default is no indentation)
4.4.2.4.4 specify pattern searches are wrapped around end of file (default)
4.4.2.4.5 specify command mode is prompted (default)
4.4.2.4.6 print all output lines with line numbers (default is no line numbers)
4.4.2.4.7 specify Ada - changes definition of word (full screen mode) to be Ada lexical unit, paragraph is to be delimited by matching pairs of reserved words (begin-end,loop-end loop, if-end if, etc) (default is no Ada)
4.4.2.5 edit scripts - use of "canned" edit commands out of a file

5.0 USER OUTPUT
5.1 Diagnostics
5.1.1 Ada Compilation Errors
5.1.1.1 syntax error reporting (severity/description)
5.1.1.2 pre-semantic error reporting (severity/description)

5.1.1.3 semantic error reporting (severity/description)
5.1.1.4 storage error reporting (severity/description)
5.1.2 Error reporting through MAPSE Command Processor
5.2 Listings
5.2.1 Ada source text listing
5.2.2 assembly language listing
5.2.3 cross reference listing of identifiers
5.2.4 symbol table listing
5.3 Text
5.4 Tables
5.5 Graphics
5.6 On-Line Assistance
5.6.1 Command Assistance
5.6.1.1 help feature in MAPSE Command Processor
5.6.1.1.1 help with no parameters - general help description
5.6.1.1.2 help with parameters - description of specific AIE feature
5.6.2 Error Assistance
5.6.3 On-Line Tutor
5.6.4 Definition Assistance
5.6.5 Menu Assistance

6.0 MACHINE OUTPUT
6.1 Object Code
6.1.1 IBM 4341 Object Code generation

7.0 SUBJECT INPUT
7.1 Text Input
7.2 Data Input
7.3 Code Input
7.3.1 Ada Code Input
7.4 VHLL Input
7.4.1 Specifications

8.0 CONTROL INPUT
8.1 Parameters
8.2 Commands
8.2.1 external program invocation
8.2.1.1 function, procedure call, script invocation
8.2.1.1.1 control commands
8.2.1.1.1.1 stop execution of program
8.2.1.1.1.2 start execution of program
8.2.1.1.1.3 cancel execution of program
8.2.1.1.1.4 determine status of executing program
8.2.1.1.2 context object attributes
8.2.1.1.2.1 terminated - indicates whether execution is completed
8.2.1.1.2.2 execution_time - total execution time
8.2.1.1.2.3 exit_status - "ok", "cancelled", "interrupted"
8.2.2 expression manipulation
8.2.2.1 assignment (:=)
8.2.2.2 get - read an arbitrary sequence of literals

8.2.2.3 put - print the values of an arbitrary sequence of expressions

8.2.3 database manipulation

8.2.3.1 Simple Objects

8.2.3.1.1 copy a file/simple object

8.2.3.1.2 delete a file/simple object

8.2.3.1.3 rename a file/simple object

8.2.3.1.4 create a file/simple object

8.2.3.1.5 open a file/simple object

8.2.3.1.6 close a file/simple object

8.2.3.1.7 write a file/simple object

8.2.3.1.8 read a file/simple object

8.2.3.2 Categories

8.2.3.2.1 create category template object

8.2.3.2.2 define an attribute to reside within an extended object

8.2.3.2.3 specify that an attribute will be constant

8.2.3.2.4 determine if an attribute is a constant

8.2.4 control commands

8.2.4.1 if - select for execution one or none of a sequence of commands

8.2.4.2 loop - sequence of commands executed zero or more times

8.2.5 termination

8.2.5.1 return - return control to MCP invoker

8.2.5.2 logout

8.2.5.3 suspend - terminate MCP processing, but maintain current context

8.2.6 shutdown commands - the user may specify a sequence of commands that are to be performed when a logout or return is encountered.

8.2.7 resume (a previously suspended MCP session)

8.2.8 I/O redirection

8.2.8.1 -< - redirects a command's standard input

8.2.8.2 -> - redirects a command's standard output

8.2.8.3 -¦ - between two commands indicates that the commands are to be connected via a pipe

8.2.9 background execution of commands

8.2.10 compound commands - block structure (begin .. end)

8.2.11 MCP variables

8.2.11.1 variable attributes (type, integer, real, string, boolean, array, record, length)

8.2.12 predefined variables

8.2.12.1 %ENVIRONMENT - control MCP options

8.2.12.1.1 components:

8.2.12.1.1.1 prompt - defines the MCP user prompt

8.2.12.1.1.2 inform_default_out - user informed of each default OUT parameter (initial value is false)

8.2.12.1.1.3 auto_redefine - default OUT parameter replaces an already existing variable with same name (initial value is false)

8.2.12.2 %STATUS - placeholders for values generated during command processing

8.2.12.2.1 components:
8.2.12.2.1.1 fcontexts - contains names of all context objects within last foreground command
8.2.12.2.1.2 execution_time - contains execution time of last completed foreground command
8.2.12.2.1.3 exit_status - contains the exit status of last completed foreground command
8.2.12.2.1.4 last_task - contains the name of the last background task
8.2.12.2.1.5 active_tasks - contains a list of all currently executing tasks
8.2.13 exec - takes a string parameter and executes it as an MCP command
8.2.14 renames command
8.2.15 nested MCP's
8.2.16 login
8.3 Command Procedures
8.3.1 scripts - sequence of MCP commands stored in a database object
8.3.1.1 parameters - both in an out
8.4 Pipes
8.4.1 -¦ - between two commands indicates that the commands are to be connected via a pipe

A.2 ADA LANGUAGE SYSTEM (ALS) DESCRIPTION

A.2.1 The ALS environment database stores all information relevant to a software project and is a self-contained entity independent of the host file system. Information to be stored includes Ada and Assembly language source text, machine code representations of programs, test data, log files, statistics, documentation and relationships among programs and compilation units. The information is accessed through the ALS Database Manager.

A.2.1.1 The database is organized as a directed acyclic graph (DAG). Objects in the database, called nodes, have properties called attributes and associations. Attributes are named properties with character string values that enable the database to contain character string information about a node. Associations are named properties with values that are collections of pointers to other nodes that enable the database to contain arbitrary networks of relationships among nodes. There are three basic types of nodes, files, directories, and variation headers. Directories and variation headers are nodes in the DAG and files are leaves within the structure. File nodes contain a data portion that may be read and written by Ada programs through the Ada I/O packages (INPUT_OUTPUT and TEXT_IO packages). Directory nodes name and group other nodes. When a node is created, it is created "within" a directory called the parent directory. Every node except the root has exactly one parent directory and contains information identifying the directory that is its parent. As well, each directory contains a specification of the nodes grouped in it.

A.2.1.2 The tracking of changes is supported through the use of file revisions. These revisions are a linearly ordered set of numbered files. Upon creation of a file, the revision number one is automatically assigned. The only way to create a new revision is to use the "revise" tool. Only the most recent revision may be modified (i.e., earlier revisions may only be read and executed).

A.2.1.3 The environment database supports the collection of related objects that are equal alternatives and do not supersede one another (e.g., the collection of bodies that may implement the same Ada package specification). These collections, called variation sets, are unordered sets of named nodes with a header node called the variation set header. The header node is the parent of each variation in the set. Directories may contain variation sets as offspring and variation sets may contain directories as offspring.

A.2.1.4 Program libraries contain the information necessary to support separate compilation, perform partial and complete link-edits of Ada programs, and incorporate subroutines written in languages other than Ada. As well, program structure is specified to support analysis and debugging requirements. Files within a program library are called containers and include specification of externally visible Ada names, statistics, object code, etc. Assemblers, linkers and importers each create a single container. When a unit is recompiled, a new revision

of its container is created (the old container is not replaced by the new). A full history of successful compilations is maintained in each program library.

A.2.1.5 Access control within the ALS database is handled through attributes and user names (e.g., team.member). Every node has a set of attributes specifying the access controls. Initially, when a node is created, any member of the creator's team may read and execute the contents of the node. Access control attributes may then be set using the "chattr" tool.

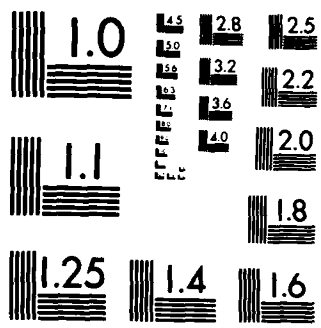
A.2.2 The Ada compiler is divided into two major phases, the machine independent section (MI) and the code generator targetted to the VAX 11/780(VMS). The machine independent portions include compiler control, Ada language program analysis (front end), machine independent optimizations, utility support, and tracing support. The control function includes receiving the user's invocation, managing the flow of control through the compilation, generating listings, saving results and terminating the compilation when encountering a fatal error. The machine independent optimizations performed include redundant constraint check elimination, constant expression evaluation, common subexpression elimination, code motion, dead code elimination, etc. Within the utility support, statistics are gathered about each compilation which may be printed out later. These statistics include the number of compilation units, the number of lexical units of each type, the number of source input records, and optimization statistics. The output of the MI section is the DIANA intermediate language which is a modified version of the Tartan Laboratories DIANA and different still from the Intermetrics DIANA. The code generator portion of the Ada compiler includes two phases, translation and selection. Translation is the mapping of a user program, which is in terms of Ada objects, operators and control functions, into an equivalent set of machine-level data objects, operators and control functions (i.e., replacing generics, in-line functions and declarations with explicit code which is still in the DIANA form and target independent). Selection is the process of determining the sequences of VAX 11/780 instructions that correctly implement the user program.

A.2.3 The ALS Linker consists of two parts, a linking tool and an exporter. The linking tool combines separately created containers which may be Ada programs compiled on the ALS Ada compiler, assembly language programs assembled on the VAX 11/780 assembler, or previously linked modules. The exporter is the tool which makes a linked program into a VAX 11/780 load module.

A.2.4 The File Administrator and Configuration Control Tools are two features of the ALS which provide operating system level support. The File Administrator is a collection of tools providing services for comparing elements in the database, balancing disk and space requirements, backup and recovery of the ALS upon system failure or human error, long term storage, and ALS-to-ALS database transmission. The Configuration Control Tools include functions to manipulate the program library (create, delete, interrogate, etc.).

A.2.5 The command language (CL), called an ALS session, provides a single and uniform interface between the user and tools. The CL is interpreted by the command language processor (CLP) which is used to invoke other tools as well as instances of itself. A session is divided into a series of command streams which begin when the user logs on the ALS (from VMS) and each time a command procedure is invoked. A command stream ends when the user logs off the ALS and each time a command procedure ends. A command procedure is a sequence of commands (terminated with a line mark or semicolon) stored in the database.

AD-A153 609 EVALUATION AND VALIDATION (E&V) TEAM PUBLIC REPORT 3/6
VOLUME 1(U) AIR FORCE WRIGHT AERONAUTICAL LABS
WRIGHT-PATTERSON AFB OH V L CASTOR 30 NOV 84
UNCLASSIFIED AFWAL-TR-85-1016-VOL-1 F/G 14/2 NL


1.0
2.8
2.5
3.2
2.2
3.6
4.0
2.0
1.1
1.8
1.25
1.4
1.6

A.2.6 ALS TAXONOMY

1.0 MANAGEMENT
- 1.1 Configuration Control
  - 1.1.1 Attribute Functions
    - 1.1.1.1 set an attribute
    - 1.1.1.2 get an attribute
  - 1.1.2 Association Functions
    - 1.1.2.1 get an association
    - 1.1.2.2 add a reference
    - 1.1.2.3 change a reference
    - 1.1.2.4 delete a reference
  - 1.1.3 Access Control
    - 1.1.3.1 get access list for user
    - 1.1.3.2 set access list for user
    - 1.1.3.3 set access list for team
  - 1.1.4 Archiving and History Functions
    - 1.1.4.1 get history of a file
    - 1.1.4.2 create a new archive source
    - 1.1.4.3 create a new version of archived source
    - 1.1.4.4 get most recent version of archived source
    - 1.1.4.5 get a specified version of archived source
- 1.2 Information Management
  - 1.2.1 Ada Library Management
    - 1.2.1.1 Program Library Manager
      - 1.2.1.1.2 Library Unit Functions
        - 1.2.1.1.2.1 create a unit
        - 1.2.1.1.2.2 delete a unit
        - 1.2.1.1.2.3 copy a unit
        - 1.2.1.1.2.4 share a unit
  - 1.2.2 Specification Management
  - 1.2.3 Data Dictionary Management
  - 1.2.4 Ada Package Management
  - 1.2.5 Test Management
- 1.3 Project Management
  - 1.3.1 Cost Estimation
  - 1.3.2 Scheduling
  - 1.3.3 Tracking
- 1.4 System Management
  - 1.4.1 Backup and Recovery
    - 1.4.1.1 full backup
    - 1.4.1.2 increment backup
    - 1.4.1.3 tree backup
    - 1.4.1.4 list of all node changes since last backup

2.0 STATIC ANALYSIS
- 2.1 Type Analysis
  - 2.1.1 syntax and semantic analysis
- 2.2 Interface Analysis
- 2.3 Statistical Profiling
  - 2.3.1 statistics listing
- 2.4 Cross Reference
  - 2.4.1 cross reference listing

2.5 Auditing
2.6 Complexity Measurement
2.7 Completeness Checking
2.8 Consistency Checking
2.9 Structure Testing
2.10 Reference Analysis

3.0 DYNAMIC ANALYSIS
3.1 Timing Analysis
3.2 Tuning Analysis
3.3 Tracing/Debugging
3.3.1 Debugging Capabilities
3.3.1.1 Execution Control
3.3.1.1.1 start execution of program
3.3.1.1.2 stop execution of program
3.3.1.2 Information Commands
3.3.1.2.1 display value of specified global variable
3.3.1.2.2 display currently defined substituters and their values
3.3.1.2.3 display current time
3.3.1.2.4 display current date
3.3.1.3 DEBUG Control in the CLP
3.3.1.3.1 show each line as it is about to be processed
3.3.1.3.2 show each command after substitution has taken place
3.3.1.3.3 show each command after expression evaluation has been performed
3.3.1.3.4 show the initial values of parameter substituters for command procedures

4.0 TRANSFORMATION
4.1 Formatting
4.1.1 Command Line Functions
4.1.1.1 use the backspace characters
4.1.1.2 enable the boldface function
4.1.1.3 enable appearance of change bars in output
4.1.1.4 create a table of contents
4.1.1.5 trace the operation of defined format commands
4.1.1.6 specify number of blank lines at top of each page
4.1.1.7 number of lines per output page
4.1.1.8 create an index file
4.1.1.9 write termination message to terminal
4.1.1.10 specify where to display error messages
4.1.1.11 flagged text underlined by nonspacing character
4.1.1.12 specify where output should go
4.1.1.13 specify a group of pages to be output
4.1.1.14 pause after printing each page of output
4.1.1.15 cause text on each page to be shifted to right
4.1.1.16 underline with separate characters on next line
4.1.1.17 output line numbers from input file
4.1.1.18 use line feed to advance to top of page
4.1.1.19 specify character for normal underlining
4.1.1.20 execution of conditional commands

4.1.2 Flag Control Commands

4.1.2.1 treat next character as ordinary text
4.1.2.2 make next character boldface
4.1.2.3 break word here if at end of line
4.1.2.4 capitalize all of next word
4.1.2.5 beginning of a comment
4.1.2.6 start of formatter command
4.1.2.7 hyphenate a word if end of line
4.1.2.8 index the next word
4.1.2.9 make next character lower case
4.1.2.10 overstrike previous character with next character
4.1.2.11 insert extra interword space after character
4.1.2.12 insert unexpandable space
4.1.2.13 subindex next word or phrase in INDEX command
4.1.2.14 insert date or time
4.1.2.15 underline the next character
4.1.2.16 make next character upper case

4.1.3 Formatter Commands

4.1.3.1 specify beginning of appendix, identifying letter, and title
4.1.3.2 automatically issue JUSTIFY
4.1.3.3 automatically paragraph
4.1.3.4 use header level titles for running subtitles
4.1.3.5 start new paragraph for each line that doesn't start with tab or a space
4.1.3.6 insert number of blank lines that are specified
4.1.3.7 end current line immediately
4.1.3.8 center line of text around character or line
4.1.3.9 specify beginning of chapter,number it,and title
4.1.3.10 accept control characters as normal text
4.1.3.11 put current date in running header
4.1.3.12 specify sequential lettering of appendices
4.1.3.13 specify form of sequential numbers of chapters
4.1.3.14 specify sequential numbering of items a list
4.1.2.15 specify form of sequential numbers of section headers
4.1.2.16 specify form of sequential numbers of pages
4.1.2.17 specify form of sequential lettering of subpage characters
4.1.2.18 put alphabetized index into single column form on a new page
4.1.2.19 control insertion of vertical bars at start of text
4.1.2.20 allows index flag and qualifier
4.1.2.21 enable collection of data for table of contents
4.1.2.22 create index without page number reference
4.1.2.23 leave room at top of page to insert a figure
4.1.2.24 same as above, but if not enough room end page immediately and start at top of next page
4.1.2.25 treat line ending like spaces
4.1.2.26 running head appears on first page of document without chapters
4.1.2.27 place text at bottom of current page
4.1.2.28 specify head on header level

4.1.2.29 print one or two lines of information at top of page
4.1.2.30 specify case of word page that proceeds the page number
4.1.2.31 determines if a portion of a file is processed, according to conditions set by user
4.1.2.32 first line of text to begin at specified position
4.1.2.33 create index entry with page number reference
4.1.2.34 insert spaces between words to make them reach right hand margin
4.1.2.35 keep blank lines from input file in output file
4.1.2.36 rearrange running head on pages
4.1.2.37 set left margin to specify position
4.1.2.38 specify beginning of a list
4.1.2.39 specify beginning of each element in a list
4.1.2.40 format text exactly as typed
4.1.2.41 insertion of the space for only one line of text
4.1.2.42 highlight portion of text
4.1.2.43 specify letter with which appendixes begin
4.1.2.44 specify number with which chapters will begin
4.1.2.45 specify beginning number of headers
4.1.2.46 specify number with which items will begin
4.1.2.47 specify beginning of a new number sequence
4.1.2.48 specify beginning of running page numbers
4.1.2.49 specify beginning of subpage numbers
4.1.2.50 start a new page
4.1.2.51 specify maximum number of lines per page and maximum number of characters per line
4.1.2.52 enable paging
4.1.2.53 specify spacing and page placement when paragraphing
4.1.2.54 restore insertion of extra space after .,;,:,?,!
4.1.2.55 put alphabetized index in column format
4.1.2.56 specify characters to be repeated
4.1.2.57 process several format files at the same time and merge them into one output file
4.1.2.58 position a single line of text relative to right margin
4.1.2.59 set the right margin
4.1.2.60 insert into table of contents, file commands, files, and text
4.1.2.61 specify current date
4.1.2.62 specify current time
4.1.2.63 preset level of next section head
4.1.2.64 specify values for paragraph
4.1.2.65 insert blank lines specified by SPACING
4.1.2.66 set amount of spacing between lines of text
4.1.2.67 change format of levels of section heads
4.1.2.68 begin a new page and a new format of page numbering
4.1.2.69 specify a subtitle for a running head
4.1.2.70 change the current position of tab stops
4.1.2.71 specify the amount of text on a single page
4.1.2.72 specify the title for a running head

4.1.2.73 identifies which conditional commands will be processed

4.2 Optimization

4.3 Compilation

4.3.1 Listing Control Options

4.3.1.1 SOURCE - produce a source listing

4.3.1.2 REFORMAT - reformat the source

4.3.1.3 LIST INCLUDE - list text brought in by include

4.3.1.4 PRIVATE - list text in private part of package specification

4.3.1.5 NOTES - include diagnostics of severity NOTE in listing

4.3.1.6 ATTRIBUTE - produce a symbol attribute listing

4.3.1.7 XREF - produce a cross reference listing

4.3.1.8 STATISTICS - produce a statistics listing

4.3.1.9 MACHINE - produce a machine code listing

4.3.1.10 DIAGNOSTICS - produce a diagnostics summary listing

4.3.2 Maintenance Aid Options

4.3.2.1 COMPILER_DEBUG - permits maintenance options to have effect

4.3.2.2 SAVE_CONTAINER_nn - save state of container in a temporary file

4.3.2.3 USE_CONTAINER_nn - use container stored in temporary file

4.3.2.4 FLAG_nn_string - specify options which have effect on specified portion of compilation

4.3.2.5 maintenance_nn - maintenance and traces should apply to this statement

4.3.2.6 STATEMENT_RANGE_nnn_nnn - maintenance and traces should apply to this range of text

4.3.2.7 ID_identifier _ trace information about this identifier

4.3.2.8 STANDARD_COMPILE - compile a new version of package STANDARD

4.3.3 Other Compiler Options

4.3.3.1 CODE_ON_WARNING - generate code when there are diagnostics of severity warning

4.3.3.2 CONTAINER_GENERATION - produce a container

4.3.3.3 OPTIMIZE - optimize in accordance to optimize in text

4.3.3.4 FREQUENCY - count how often things are executed

4.3.3.5 TRACE_BACK - listing of all subprograms executing when exception raised

4.4 Editing

4.4.1 Syntax Directed

4.4.2 Basic Editor

4.4.2.1 Key Pad Functions

4.4.2.1.1 move cursor to top of buffer

4.4.2.1.2 move cursor to bottom of buffer

4.4.2.1.3 move cursor left/right one character/word

4.4.2.1.4 move cursor up/down one character/word/line

4.4.2.1.5 move text to a temporary buffer

4.4.2.1.6 restore text in buffer to original location

4.4.2.1.7 delete a character/word/line
4.4.2.1.8 restore deleted character/word/line
4.4.2.1.9 change the case of character/string
4.4.2.1.10 insert character/word/line/page
4.4.2.1.11 locate a string above/below current position
4.4.2.1.12 locate next occurrence of string above/below position
4.4.2.1.13 delete edit buffer
4.4.2.1.14 move text from buffer to file
4.4.2.2 Line Editing Functions
4.4.2.2.1 all of key pad functions, plus ability to specify a range of text
4.4.2.2.2 copy a line from one buffer to another
4.4.2.2.3 copy a specified range of text into a file
4.4.2.2.4 resequence line numbers
4.4.2.2.5 display a specified range of text
4.4.2.3 Non Key Pad Functions
4.4.2.3.1 all of key pad functions

5.0 USER OUTPUT
5.1 Diagnostics
5.1.1 Error reporting through the ALS
5.1.1.1 FATAL - user/internal error reporting of highest severity (statement number/severity/code/description)
5.1.1.2 SYSTEM - internal error reporting (satement number/severity/code/description)
5.1.1.3 ERROR - user/inernal error reporting (statement number/severity/code/description)
5.1.1.4 WARNING - possible unintended results by user (statement number/severity/code/description)
5.1.1.5 NOTE - reporting of unusual action taken (statement number/severity/code/description)
5.2 Listings
5.2.1 Ada source text listing
5.2.2 symbol attribute listing
5.2.3 cross reference listing
5.2.4 compilation statistics listing
5.2.5 machine code listing
5.2.6 diagnostics summary listing
5.2.7 compilation summary listing
5.3 Text
5.4 Tables
5.5 Graphics
5.6 On_Line Assistance
5.6.1 Command Assistance
5.6.1.1 help feature in command processor
5.6.1.1.1 help with no parameter - information about HELP command
5.6.1.1.2 help with parameter - information about specific subject
5.6.1.1.3 quick help - single piece of information about specified subject

```
      5.6.1.2   help feature of EDT editor
        5.6 1.2.1   help with no parameter - information about
                    HELP facility
        5.6.1.2.2   help with parameter - information about
                    specific subject
    5.6.2   Error Assistance
    5.6.3   On_Line Tutor
    5.6.4   Definition Assistance
    5.6.5   Menu Assistance

6.0   MACHINE OUTPUT
  6.1   Object Code
    6.1.1   VAX - 11/780 object code

7.0   SUBJECT INPUT
  7.1   Text Input
  7.2   Data Input
  7.3   Code Input
    7.3.1   Ada code input
  7.4   VHLL input

8.0   CONTROL INPUT
  8.1   Parameters
  8.2   Commands
    8.2.1   function, procedure calls, script invocation
      8.2.1.1   control commands
        8.2.1.1.1   start execution of a program
        8.2.1.1.2   stop execution of a program
    8.2.2   database manipulation
      8.2.2.1   file manipulation
        8.2.2.1.1   copy a file
        8.2.2.1.2   delete a file
        8.2.2.1.3   create a file
        8.2.2.1.4   rename a file
      8.2.2.2   Variation Set Functions
        8.2.2.2.1   create a variation set
        8.2.2.2.2   delete a variation set
    8.2.3   termination
      8.2.3.1   exit - leave ALS  environment
      8.2.3.2   logout
      8.2.3.3   suspend - stop ALS session
    8.2.4   login
    8.2.5   control commands
      8.2.5.1   if - execute a sequence of one or more
                     commands depending on a condition
      8.2.5.2   null - process next command
      8.2.5.3   loop - repeat execution of a sequence of commands
                     in a loop body
      8.2.5.4   loop with while - evaluate and test condition
                                  before executing loop body
      8.2.5.5   exit - terminate enclosing loop
    8.2.6   I/O redirection
      8.2.6.1   redirect a commands standard input
      8.2.6.2   redirect a commands standard output
```

8.2.7 background execution of commands
8.2.8 nested CLP commands
8.2.9 CLP substitutors
8.2.10 predefined substitutors
8.2.10.1 #STATUS
8.2.10.1.1 indicate success or failure of invoking command
8.2.10.1.2 indicate success or failure of execution of program or command procedure
8.2.10.1.3 information about executed program or command procedure
8.3 Command Procedures
8.3.1 create a command procedure
8.3.2 invoke a command procedure
8.3.3 pass parameter in an out of procedure
8.4 Pipes

A.3 ADA LANGUAGE SYSTEM/NAVY DESCRIPTION

A.3.1 The ALS/N system will be implemented as extensions to the Army's Ada Language System. The ALS/N has nine functional areas broken down into two catagories. Catagory one is called the Minimal Ada Programming Support Environment (MAPSE) and includes as the functional areas the Language Processor, the Separate Compilation Support, the Code Manipulation, the Machine Transportable Support Software (MTASS) Interface, the Text Manipulation, the User Access Support, and the MAPSE Run-Time Environment. The MTASS system currently support the AN/UYK-43, AN/UYK-7, AN/UYK-20, AN/UYK-44 and AN/AYK target computers. The other catagory is known as the Run-Time Environment (RTE) consisting of the Runtime Operating System (RTOS) and the Run-Time Application Support (RTAS).

A.3.2 The MAPSE catagory software will execute within the host computer (initially VAX/VMS). The RTE catagory will run on Navy standard computers (termed "embedded target computers"). The RTE will provide support for the target computers and will not attempt to provide host-like services.

A.3.3 The ALS/N supports two classes of target computers. The term "Ada/L" refers to AN/UYK-43 (32 bit computer) and "Ada/M" refers to support of the AN/UYK-44 and AN/AYK-14 (16 bit computers).

A.3.4 The MAPSE catagory will consist of Ada/L and Ada/M Code Generators, Ada/L and Ada/M Linkers, Ada/L and Ada/M Embedded Target Listing Tools, Ada/L and Ada/M Importer, Ada/L and Ada/M exporter, and Ada/L and Ada/M Target Debugger. The RTE catagory will consist of Ada/L and Ada/M Run-Time Executives, Ada/L and Ada/M Run-Time Support Library, Ada/L and Ada/M Run-Time Loaders, Ada/L and Ada/M Run-Time Debuggers and Ada/L and Ada/M Run-Time Performance Measurement Aids. Included as a Program Support Environment is Configuration Management Identification Tools, General Purpose Text Editor, Text Formatter, Report Generator, Common Interface Routine, Interhost Telecommunications Interface and Embedded Target Computer Interface.

APPENDIX B

REFERENCES

[1] Buxton, J.N., Requirements for Ada Programming Support Environment (APSE) - -"STONEMAN", U.S. Department of Defense, February 1980.

[2] Houghton, R., A Taxonomy of Tool Features for the Ada Programming Support Environment (APSE), U.S. Department Commerce, National Bureau Of Standards, December 1982, Issued February 1983.

[3] Intermetrics, Inc., "System Specification for Ada Integrated Environment", 12 November 1982.

[4] Intermetrics, Inc., "Computer Program Development Specification for Ada Integrated Environment: KAPSE/Database", 12 November 1982.

[5] Intermetrics, Inc., "Computer Program Development Specification for Ada Integrated Environment: Ada Compiler Phases", 5 November 1982.

[6] Intermetrics, Inc., "Computer Program Development Specification for Ada Integrated Environment: DIANA", 23 December 1982.

[7] Intermetrics, Inc., "Computer Program Development Specification for Ada Integrated Environment: Program Integration Facilities", 4 August 1983.

[8] Intermetrics, Inc., "Computer Program Development Specification for Ada Integrated Environment: MAPSE Command Processor", 1 December 1982.

[9] Intermetrics, Inc., "Computer Program Development Specification for Ada Integrated Environment: MAPSE Debugging Facilities", 5 January 1983. [10] Tartan Laboratories, "DIANA Reference Manual", Revision 3, 28 February 1983.

[11] SofTech, Inc., Ada Language System Specification, Volumes I and II, June 1981.

[12] SofTech, Inc., Ada Language System Compiler Machine Independent Section B5 Specification, February 1982.

[13] SofTech, Inc., Ada Language System VAX 11/780 Code Generator B5 Specification, January 1982.

[14] SofTech, Inc., Ada Language System VAX 11/780 Linker B5 Specification, June 1981.

[15] SofTech, Inc., Ada Language System Configuration Control Tools B5 Specification, December 1981.

[16] SofTech, Inc., Ada Language System File Administrator B5 Specification, January 1982.

[17] SofTech, Inc., Ada Language System Textbook, 28 November 1983.

[18] Evaluation and Validation (E&V) Plan, Version 1.0, 30 November 1983.

APPENDIX C

DEFINITIONS

ASSOCIATIONS - Named properties of a node with values that are collections of pointers to other nodes.

CATALOG - A catalog provides a means of accessing a set of logically connected compilation units which may or may not comprise an entire program library. There are two kinds of catalogs: a primary catalog, analogous to a working directory, providing access to the library units currently under development by a particular programmer or group of programmers; and a resource catalog, analogous to the traditional notion of a library, providing access to units which might be used as utility routines or "resources" by other library units. A primary may be frequently changed, whereas, a resource catalog is relatively stable.

COLLECTION - A collection is a set of database objects which provide a convenient unit for resource allocation, access control, and revision maintenance. Catalogs are represented as objects within a collection.

CONTAINER - A file within a program library. It includes specifications of externally visible Ada names, statistics, object code, and other information. One container is produced for each compilation unit compiled.

DOMAIN - A domain is a collection of one or more separate data structures (VMSDs or subdomains) that may reference one another.

PARTITION - A partition is a set of composite objects "grouped together" according to their distinguishing and non-distinguishing attributes.

REP ANALYZER - The Rep Analyzer combines the definitions for one or more virtual record types and enforces the restrictions and conventions required by VMM. The Rep Analyzer generates a new Ada package specification and body for each virtual record type (called the virtual record type declaration); and a package specification and body called the access package that provides access to other package templates that provides operations to manipulate the defined data structures.

ROLE - A role represents a logical set of participants allowing possible access and manipulation of extended objects.

ROLE MODIFIER - Role modifiers are predefined for all extended objects. They determine extra rights and limitations associated with the roles held. Some include: _OWNER, _READ_ONLY, and _OVERSEER.

SUBDOMAIN (VMSD) - A single VMM data structure is a directed graph

constituting a subdomain (VMSD) that can, in several contexts, be accessed as a single entity. Each VMSD contains VMM objects whose types are defined in one package specification input to the Rep Analyzer.

SUBSTITUTERS - Substitute identifiers in the command language used to denote string substituters. When appearing in a command, it is replaced by the value of the string for which it stands for.

VARIATION SET - A set containing different versions of an object, each element co-exists with the others.

VIRTUAL MEMORY METHODOLOGY - The VMM (Virtual Memory Methodology) subsystem is a tool for creating and manipulating abstract data structures (attributed directed graphs, in particular) in a machine-independent manner. A virtual memory paging scheme makes the size of any data structure independent of memory constraints of any particular hardware configuration.

VMM LOCATOR - A VMM locator is a reference to virtual records (VMM objects) that reside in other structured VMSDs. A VMM locator is the only means of consistently designating a VMM object. A locator is similar to an Ada access value since it is a typed pointer with values generated by allocation operations, and a distinguished null value which designates no object at all. The differences between locators and Ada access values are that a VMM locator value generated by an allocation during program activation can be written to an external file and then be read by a subsequent program activation and still be guaranteed to designate the same VMM object. Also, the addressing range of a VMM locator is defined by the implementation of the VMM package and is not dependent on host machine characteristics or the size of the run-time heap available to the program activation.

VMM ROOT LOCATOR - A root locator is a distinguished VMM locator which may be explicitly set or examined by VMM operations.

VIRTUAL RECORD NOTATION - Virtual record notation is data in human-readable form.

WINDOWS - Windows specify the roles (and role modifiers) to be used within an extended object in terms of the roles used in the extended object enclosing the window. There are two kinds of windows: A primary window links an extended object to its enclosing composite object. A secondary window allows an object to be viewed from a location other than the

APPENDIX D

ACRONYMS

AIE - Ada Intergrated Environment

AJPO - Ada Joint Program Office

ALS - Ada Language System

APSE - Ada Programming Support Environment

BILL - But It's Low Level - Intermetrics developed Low Level Language

CL - Command Language

CLI - Command Language Interpreter

CLP - Command Language Processor

CMS - IBM 4341 operating system

DAG - Direct Acyclic Graph

DIANA - Descriptive Intermediate Attributed Notation for Ada

DoD - Department of Defense

EDT - VAX 11/780 editor

E&V - Evaluation and Validation

KAPSE - Kernel Ada Programming Support Environment

LEXSYN - Lexical and Syntatic phase of AIE compiler

MAPSE - Minimal Ada Programming Support Environment

MCL - MAPSE Command Language

MCP - MAPSE Command Processor

MI - Machine Independent (section of ALS compiler)

OS/VS1 - IBM 4341 operating system

PIF - Program Interface Facility

UTS - Amdohl developed UNIX-like operating system

VM - Virtual Memory - IBM 4341 set of low level routines

VMM - Virtual Memory Methodology

VMS - Vax 11/780 operating system

VMSD - Virtual Memory Subdomain

APPENDIX E

APSEWG MEMBERS

Elizabeth Kean (CHAIR)
RADC/COES
Griffiss AFB, NY

Gina Burt (VICE CHAIR)
AFALC/PTEC
Wright-Patterson AFB OH

Terry Humphrey
Johnson Space Center
Houston, TX

Mars Gralia
John Hopkins University
Laurel, MD

Guy Taylor
FCDSSA
Virginia Beach, VA

Georgeanne Chitwood
ASD/ADOL
Wright-Patterson AFB OH

Capt Albert Deese
ASD/ADOL
Wright-Patterson AFB OH

Doug Yarborough
GTE Government Systems
1 Federal Street
Billerica, MA

William Grabowski
GTE Government Systems
1 Federal Street
Billerica, MA

DISTINGUISHED REVIEWERS:

Paul Reilly
Data General Corporation
4400 Computer Drive
Westboro, MA

Bard Crawford
TASC
1 Jacob Way
Reading, MA

Marlow Henne
Harris Corporation
150 Wikham Rd
Melbourne, FL

APPENDIX D

EVALUATION and VALIDATION
TECHNICAL COORDINATION STRATEGY DOCUMENT
VERSION 1.0
28 AUGUST 1984

Table of Contents

Table of Contents (Continued)

Table of Contents (Continued)

Table of Contents (Continued)

Table of Contents (Continued)

1. INTRODUCTION

1.1 Objective of the Technical Coordination Strategy Document

The objective of the Technical Coordination Strategy Document (TCSD) is to provide a mechanism whereby both Department of Defense (DoD) and contractor technical efforts/organizations which are potentially related to the Evaluation and Validation (E&V) of Ada Programming Support Environments (APSEs) Task, may be identified. Specifically, the TCSD will : a) identify related technical efforts; b) identify relationships between the E&V Task and each related effort; c) identify areas of mutual benefit; d) identify impact of schedules; e) identify the level of coordination required between the E&V Task and each related effort; and f) identify issues which require resolution with respect to the mutual benefit of both the E&V Task and the particular related effort involved.

1.2 Background

The purpose of the E&V Task, which is sponsored by the Ada Joint Program Office (AJPO), is to develop the technology by which APSEs will be evaluated and validated. The term "evaluation" represents a qualitative assessment of an APSE component for which no objective standard exists. The term "validation" represents a quantitative measurement of an APSE component for which both a standard and metrics exist. Techniques and tools will be developed which will provide a capability to perform assessment of APSEs and to determine conformance of APSEs to the Common APSE Interface Set (CAIS), which is being developed by the Kernel Ada Programming Support Environment (KAPSE) Interface Team (KIT) and their companion organization, the KAPSE Interface Team from Industry and Academia (KITIA).

As the E&V technology is developed, it will be made available to the user community for implementation by the DoD components, industry, and academia as appropriate.

2. SCOPE

The overall goal of the TCSD is to establish lines of communication between the E&V Task and other related DoD and industry efforts/organizations. It is essential to the success and effectiveness of the E&V Task as a whole, to coordinate with other related efforts. This type of coordination and communication will keep other organizations and efforts abreast of the E&V Task and its resulting technology, and will identify those areas of evaluation and validation which are of mutual benefit. This exchange of technical information relevant to E&V will be monitored by the Technical Coordination Working Group (TECWG) and transmitted to the other E&V working groups as appropriate.

It is the responsibility of the TECWG to : 1) develop the TCSD; 2) provide technical presentations to the E&V Team on related technical efforts identified; and 3) provide position papers throughout the duration of the E&V Task which address particular aspects of the E&V Task with related tasks/efforts. Also, the TECWG is responsible for both providing and updating the status of these technically related efforts to the Team, as well as enhancing this document in future revisions with the identification of additional tasks/efforts, and updated information on currently identified efforts.

This initial version of the TCSD was developed by combining the various Technical Coordination Statements which were prepared by members of the E&V Team, who are presently involved or associated with the identified task/effort. The following represents the Technical Coordination Statement template used by each E&V Team member :

1. Name of the technically related effort
2. Purpose
3. Relationship to the E&V Task
4. Benefits to the E&V Task
5. Benefits to the related effort/organization
6. Impact on E&V Task schedules
7. Impact on related effort/organization schedules
8. Required level of coordination
9. Resolution of issues
10. Focal point

3. APPROACH

Currently, two primary methods to establish and promote coordination between the E&V Task and other related technical efforts/organizations have been identified, and are outlined below.

3.1 Invited Briefings

Invitations will be extended to particular individuals to attend the quarterly E&V Team meetings as appropriate, for the purpose of briefing specific related efforts. These briefings will provide interactive communication and dialogue between the E&V Team members and the particular briefer with respect to exchange of technical information.

3.2 Technical Coordination Statements/TECWG Briefings

Technical Coordination Statements will be used in conjunction with the method indicated in paragraph 3.1. The purpose of these statements is to identify a related effort (or organization), and elaborate upon various aspects of this relationship. Currently, ten specific relational aspects are identified on the Technical Coordination Statement, as indicated above.

In addition, the TECWG Chairperson (or Vice-Chairperson) will update the E&V Team on the status of various related technical efforts at the quarterly E&V meetings. These briefings will adhere to the following format :

NAME OF RELATED EFFORT/ORGANIZATION

PROGRAM MANAGER (ADDRESS/PHONE)

PROGRESS STATEMENT (SIGNIFICANT EVENTS/MILESTONES) SINCE LAST UPDATE

DATE

4. IDENTIFICATION/ELABORATION OF RELATED TECHNICAL EFFORTS

The following technical efforts/organizations have been identified as being related to the E&V Task, and are elaborated in the following paragraphs.

4.1 Ada C3I Test and Evaluation

4.1.1 Purpose

The purpose of this effort is to test and evaluate the effectiveness of using an integrated Ada programming environment in an Air Force command, control, and intelligence (C3I) system software development project. Areas to be evaluated include the use of Ada as a programming design language, documentation, the quality and quantity of the code produced, and compiler performance and productivity. In order to determine the effectiveness of Ada, the software for a selected Air Force system acquisition will be implemented in Ada as a parallel effort.

4.1.2 Relationship to the E&V Task

The results of this effort will be a technical report describing the results of the test and evaluation. The results of this effort may aid in the development of requirements and criteria for the evaluation and validation of APSEs.

4.1.3 Benefits to the E&V Task

The information gained as a result of this effort may assist the E&V Task in the development of APSE evaluation and validation requirements and criteria.

4.1.4 Benefits to the Related Effort/Organization

Any requirements and criteria for the evaluation of compilers developed before this contract is awarded will aid in the evaluation of the AIE Ada compiler.

4.1.5 Impact on E&V Task Schedules

No schedule impacts are currently identified.

4.1.6 Impact on Related Effort/Organization Schedules

No schedule impacts are currently identified.

4.1.7 Required Level of Coordination

At present, Elizabeth Kean is an active member of the E&V Team, and will assist in the coordination of this effort and the E&V Task.

4.1.8 Resolution of Issues

Issues identified within the E&V Task will be handled within the E&V Task. Issues identified within the Ada C3I Test and Evaluation effort will be resolved through the RADC chain of command, up to and including the AJPO.

4.1.9 Focal Point

The focal point is indicated below :

Elizabeth Kean

Rome Air Development Center

Commercial : (315) 330-4325

Autovon : 587-4325

4.2 Ada Integrated Environment

4.2.1 Purpose

The purpose of this Air Force-directed effort is to design and develop a Minimal Ada Programming Support Environment (MAPSE) including a state-of-the-art Ada compiler. The Ada compiler will be developed for rehosting and retargeting to a number of computers. The MAPSE will also consist of software tools and aids to assist programmers and project managers in the development of Ada software. Procedures for rehosting/retargeting the compiler and the MAPSE will be developed under this effort.

4.2.2 Relationship to the E&V Task

The product of this effort, a Minimal APSE, may eventually be evaluated and validated using the requirements and criteria developed under the E&V Task.

4.2.3 Benefits to the E&V Task

The AIE is the Air Force's implementation of a Minimal APSE. The AIE can be used as an aid in determining the requirements and criteria for evaluating and validating future APSEs.

4.2.4 Benefits to the Related Effort/Organization

The E&V technology developed under the E&V Task will aid in the assessment of future software tools to be incorporated in the AIE. The CAIS will eventually be implemented on the AIE. The CAIS and the CAIS validation capability will provide standardization of interfaces and a method for validating the implemented interfaces.

4.2.5 Impact on E&V Task Schedules

No schedule impacts are currently identified.

4.2.6 Impact on Related Effort/Organization Schedules

The CAIS will eventually be implemented on the AIE, therefore, the CAIS validation schedule may impact the AIE effort.

4.2.7 Required Level of Coordination

At present, Elizabeth Kean is an active member of the E&V Team and a technical evaluator on the AIE effort, and will provide coordination between this effort and the E&V Task.

4.2.8 Resolution of Issues

Issues identified within the E&V Task will be handled within the E&V Task. Issues identified within the AIE effort will be resolved through the Rome Air Development Center (RADC) chain of command, up to and including the AJPO.

4.2.9 Focal Point

The focal point is indicated below :

Donald Mark

Rome Air Development Center

Commercial : (315) 330-3398

Autovon : 587-3398

4.3 Ada Joint Program Office

4.3.1 Purpose

The purpose of the AJPO, which was established on 12 December 1980 by the Under Secretary of Defense for Research and Engineering, is to manage the DoD's effort to implement, introduce and provide life-cycle support for Ada. The AJPO must ensure the implementation and maintenance of Ada as a consistent, unambiguous standard recognized by the DoD and also by the widest possible community. The AJPO must ensure the smooth introduction and acceptance of Ada in the DoD as early as possible, consistent with the needs of individual components. The AJPO must ensure the provision of life-cycle support for Ada through the development of a robust Ada Programming Support Environment (APSE) to improve productivity both in development and in continued evolution.

4.3.2 Relationship to the E&V Task

The AJPO is the sponsor of the E&V Task. The status of the E&V Task is briefed to the AJPO at the quarterly Tri-Service review meetings.

4.3.3 Benefits to the E&V Task

The AJPO oversees all of the E&V Task activities and provides managerial direction and funding to the E&V Task as necessary.

4.3.4 Benefits to the Related Effort/Organization

The development of the E&V technology by the E&V Task supports the AJPO objective of improving the productivity in development and continued evolution of APSEs.

4.3.5 Impact on E&V Task Schedules

No schedule impacts are currently identified.

4.3.6 Impact on Related Effort/Organization Schedules

No schedule impacts are currently identified.

4.3.7 Required Level of Coordination

The AJPO focal point (LCDR Brian Schaar) attends E&V Team meetings and is on the distribution list for all E&V Team MILNET communication. In addition, the E&V Team Chairperson (Virginia Castor) is required to brief the AJPO on the status of the E&V Task at quarterly Ada Tri-Service Reviews.

4.3.8 Resolution of Issues

All such issues should be brought to the attention of the AJPO by the E&V Team Chairperson. The AJPO has final authority in the resolution of such issues.

4.3.9 Focal Point

The focal point is indicated below :

LCDR Brian Schaar

Ada Joint Program Office

RM 3D139 (400A/N DR)

The Pentagon

Washington D.C. 20301

Commercial : (202) 694-0212

Autovon : 224-0212

4.4 Ada Language System

4.4.1 Purpose

The Ada Language System (ALS) is under the direction of the U.S. Army. The purpose of this effort is to develop an APSE on the VAX/VMS 11/780 with a MIL-STD-1815A host compiler targeted to the VAX. Other targets include the Military Computer Family (MCF) Nebula instruction set architecture (ISA), and the Intel 8086.

4.4.2 Relationship to the E&V Task

The technology developed through the E&V Task can be applied to the ALS development.

4.4.3 Benefits to the E&V Task

The ALS represents the Army's implementation of an APSE. The ALS can be used as an example in determining the criteria for performing evaluation and validation on future APSEs.

4.4.4 Benefits to the Related Effort/Organization

The technology which is developed by the E&V Team will provide input to the development of the ALS. Also, the CAIS, which is currently being developed by the KIT/KITIA will be used with the ALS at a future time.

4.4.5 Impact on E&V Task Schedules

No schedule impacts are currently identified.

4.4.6 Impact on Related Effort/Organization Schedules

No schedule impacts are currently identified.

4.4.7 Required Level of Coordination

Coordination will be with the Army's ALS maintenance personnel. In addition, Mr James Williamson is currently participating as the Air Force representative on the Tri-Service ALS testing team.

4.4.8 Resolution of Issues

All such issues should be brought to the attention of the AJPO by the E&V Team Chairperson. The AJPO shall be responsible for informing the appropriate Army personnel and seeking resolution of these issues.

4.4.9 Focal Point

The focal point is indicated below :

Dennis Turner

DRSEL-TCS-Ada

U.S. Army/CECOM, Ft. Monmouth, New Jersey 07703

Commercial : (210) 544-4149

Autovon : 995-4149

4.5 Ada Test and Verification System

4.5.1 Purpose

The purpose of this effort is to design an Ada Test and Verification System (ATVS) which can be implemented as a set of computer-based software tools to improve the reliability and maintainability of Ada software systems. It is intended that this system will be applied during the coding, testing, verification, and error detection/correction phases of software development. This effort will begin with a study to determine the most advanced techniques and capabilities to be included in the design. The ATVS will then be designed as an integral component of an APSE. As a minimum, the ATVS shall be designed for use with both the Air Force's AIE and the Army's ALS.

4.5.2 Relationship to the E&V Task

The ATVS will be a portable software tool residing on an APSE. Thus, the technology developed by the E&V Task can be applied.

4.5.3 Benefits to the E&V Task

If the CAIS is available at the time of implementation, it will be used as the interface.

4.5.4 Benefits to the Related Effort/Organization

The requirements and criteria for the evaluation and validation of APSEs may result from the initial analysis study and the resulting implementation.

4.5.5 Impact on E&V Task Schedules

No schedule impacts are currently identified.

4.5.6 Impact on Related Effort/Organization Schedules

The development of the CAIS may impact the schedules of this effort.

4.5.7 Required Level of Coordination

At present, Elizabeth Kean will relay any information to and from the RADC focal point.

4.5.8 Resolution of Issues

Issues identified within the E&V Task will be handled within the E&V Task. Issues identified within the ATVS effort will be resolved through the RADC chain of command.

4.5.9 Focal Point

The focal point is indicated below :

Richard Evans

Rome Air Development Center

Commercial : (315) 330-3398

Autovon : 587-3398

4.6 Ada Validation Organization

4.6.1 Purpose

The Ada Validation Organization (AVO) is sponsored by the AJPO. Its purpose is to ensure that developers of Ada compilers have correctly implemented the standard Ada language (ANSI/MIL-STD-1815A-1983).

4.6.2 Relationship to the E&V Task

The AVO has been responsible for the development and implementation of an Ada Compiler Validation Capability (ACVC) in order to determine conformance of Ada compilers to the standard Ada language. The ACVC provides a capability to validate one particular tool within an APSE and, as such, will be incorporated within the E&V technology developed by the E&V Team.

4.6.3 Benefits to the E&V Task

The AVO has established formal procedures for validating Ada compilers and mechanisms by which the validation procedures are executed. The expertise gained through the development and implementation of these procedures will be beneficial to the E&V Team as it begins to establish recommendations for formal procedures for the implementation of E&V technology.

4.6.4 Benefits to the Related Effort/Organization

The E&V Task is responsible for developing evaluation capabilities, as well as validation capabilities. The determination of evaluation criteria for the assessment of compilers is but one of the many activities being performed by the E&V Task. The Ada compiler evaluation capability will be of particular benefit to the AVO.

4.6.5 Impact on E&V Task Schedules

No schedule impacts are currently identified.

4.6.6 Impact on Related Effort/Organization Schedules

No schedule impacts are currently identified.

4.6.7 Required Level of Coordination

The AVO Director (Thomas Probert) from the Institute for Defense Analyses (IDA) is on the distribution list to receive all E&V Team MILNET communication.

4.6.8 Resolution of Issues

Issues of concern should be coordinated through the E&V Team Chairperson and the IDA E&V focal point (John Kramer), and raised to the AJPO level if necessary for resolution.

4.6.9 Focal Point

The focal point is indicated below :

Thomas Probert

Institute for Defense Analyses

1801 N. Beauregard St.

Alexandria, Virginia 22311

Commercial : (703) 845-2517

Autovon : 289-1948 (ext. 2517)

4.7 Ada-1750A Runtime

4.7.1 Purpose

The Ada-1750A Runtime effort is under the direction of the Air Force. The purpose of the effort is to define an initial runtime model using as much industry participation as possible.

4.7.2 Relationship to the E&V Task

Part of the Ada compiler not now validated or evaluated by current techniques is the runtime executive; the out-of-line executable code that does heap management, directs hardware interrupts to Ada task ACCEPT entries, and similar book-keeping chores.

4.7.3 Benefits to the E&V Task

Since the model is Air Force owned and open to inspection, the E&V Team can determine if validating one particular runtime model is practicable, or whether evaluation only is feasible.

4.7.4 Benefits to the Related Effort/Organization

Since the CAIS may eventually incorporate a runtime module, validation will then become mandatory, but some evaluation will still be required.

4.7.5 Impact on E&V Task Schedules

No schedule impacts are currently identified.

4.7.6 Impact on Related Effort/Organization Schedules

No schedule impacts are currently identified.

4.7.7 Required Level of Coordination

At present, Mr Nelson Estes is a member of the E&V Team, as well as program manager of the Ada-1750A Runtime effort.

4.7.8 Resolution of Issues

Issues identified related to the Ada-1750A runtime model will be handled by the Air Force Ada-1750A Program Manager.

4.7.9 Focal Point

The focal point for the Ada-1750A Runtime effort is indicated below :

Nelson Estes

Embedded Computer Standardization Program Office

ASD-AFALC/AXTS

Wright-Patterson AFB, Ohio 45433

Commercial : (513) 255-5945

Autovon : 785-5945

MILNET : ESTESN@WPAFB-JALCF

4.8 Air Force Computer Resource Management Technology

4.8.1 Purpose

The over-all objective of the Air Force Computer Resource Management Technology Program Element (64740F) effort is to apply advances in computer resource management technology to the development and acquisition of Air Force and other military systems.

4.8.2 Relationship to the E&V Task

This program element (PE) supports the development and application of techniques to increase the performance and reduce the costs of mission-critical computer resources. It includes proposed programs from several Air Force Systems Command Product Divisions to develop criteria for evaluating Ada compilers. In addition, this PE includes programs which plan for and support the introduction of the Ada programming language.

4.8.3 Benefits to the E&V Task

Various 64740F-sponsored programs may result in evaluation criteria for Ada compilers and other Ada tools that may be useful to the E&V Task. Also, new software tools developed under 64740F may expand the APSE functionality definition.

4.8.4 Benefits to the Related Effort/Organization

One project within the PE is concerned with software acquisition standards and mechanisms to improve the acquisition and support of computer resources. The E&V criteria developed by the E&V Task will directly contribute to the goal of this project.

4.8.5 Impact on E&V Task Schedules

No schedule impacts are currently identified.

4.8.6 Impact on Related Effort/Organization Schedules

No schedule impacts are currently identified.

4.8.7 Required Level of Coordination

At present, Chris Anderson is an active member of the E&V Team, and the Program Planning Group (PPG) of 64740F.

4.8.8 Resolution of Issues

The focal point for coordination will assist in resolving any issues that arise which may adversely affect either effort.

4.8.9 Focal Point

The focal point is indicated below :

William Letendre

ESD/ALEE

Hanscom Air Force Base, Massachusetts 01731

Autovon : 478-5113

4.9 Common Ada Missile Packages

4.9.1 Purpose

The objective of the Common Ada Missile Packages (CAMP) program is to explore the feasibility of developing reusable software in Ada for armament applications, and an associated parts composition system. CAMP is sponsored by AFATL, the Air Force Munition and Ordnance Program Element (64602F), the STARS program, and the AJPO.

4.9.2 Relationship to the E&V Task

The product of this effort, APSE library components and associated parts composition system, may eventually be evaluated using the requirements and criteria developed under the E&V Task.

4.9.3 Benefits to the E&V Task

The reusable software components and associated parts composition system developed under the CAMP program, will result in new technology which may expand the requirements and criteria for evaluating future APSEs. In addition, the common armament functions identified by the CAMP program may serve as a basis for developing armament-specific compiler benchmarks for Ada compiler evaluation.

4.9.4 Benefits to the Related Effort/Organization

The CAIS and CAIS Validation Capability will provide standard interfaces for future APSE libraries and supporting parts composition systems, such as are being developed by the CAMP effort.

4.9.5 Impact on E&V Task Schedules

No schedule impacts are currently identified.

4.9.6 Impact on Related Effort/Organization Schedules

The following E&V Task schedules may impact the schedules of the CAMP effort.

CAIS Validation Capability contract start - 2nd Quarter FY85

Draft CVC - 1st Quarter FY86

Version 1 CVC - 1st Quarter FY86

4.9.7 Required Level of Coordination

At present, Chris Anderson is an active member of the E&V Team, and the CAMP Program Manager.

4.9.8 Resolution of Issues

Issues identified related to the E&V Task will be addressed within the E&V Task. Issues identified related to the CAMP program will be handled by the AFATL CAMP Program Manager.

4.9.9 Focal Point

The focal point is indicated below :

Chris Anderson

Air Force Armament Laboratory/DLMM

Eglin Air Force Base, Florida 32542

Commercial : (904) 882-2961

Autovon : 872-2961

4.10 Johnson Space Center Ada Project

4.10.1 Purpose

The Johnson Space Center (JSC) Testing and Analysis of DoD Ada Language Products for NASA (JSC Ada Project) effort is sponsored by National Aeronautics and Space Administration (NASA) Headquarters. The purpose of this effort is to perform testing and analyses of Ada software technology products being produced by the DoD, evaluate their applicability to future NASA projects (such as the Space Station) and develop a plan for their implementation in future NASA flight systems as a standard. The JSC Ada Project was established as a result of the Memorandum of Agreement signed in June, 1983 by Dr. Edith Martin, Deputy Under Secretary of Defense for Research and Development Technology, and Dr. Jack Kerrebrock, NASA Associate Administrator for Aeronautics and Space Technology. This agreement establishes NASA/DoD cooperation in the DoD STARS Program, and recognizes APSE Beta Testing at the JSC and the University of Houston (at Clear Lake City) as part of that cooperation.

4.10.2 Relationship to the E&V Task

Both tasks have a common goal of developing technology for use in evaluating APSEs. But in addition to using technology provided by the E&V Task, the JSC Ada Project will also develop specific evaluation criteria and tests based upon technology and tools used in current NASA spaceflight systems (e.g., the HAL/S programming support system currently used as a NASA standard).

4.10.3 Benefits to the E&V Task

The JSC Ada Project will primarily focus on use of APSEs in the development of prototype applications. Data and information from this activity will be used to develop standards and criteria later used in evaluating APSEs for NASA. This work, in conjunction with other studies and analyses at JSC, will identify additional APSE features and tools needed for support of NASA spaceflight applications projects. Information provided by the JSC Ada project should assist the E&V Task in its development of standards and criteria for use in evaluation of APSEs.

4.10.4 Benefits to the Related Effort/Organization

The technology developed by the E&V Task will be utilized in the JSC Ada Project to assist in evaluating APSEs for NASA.

4.10.5 Impact on E&V Task Schedules

No schedule impacts are currently identified.

4.10.6 Impact on Related Effort/Organization Schedules

An impact to the APSE evaluation criteria may occur.

4.10.7 Required Level of Coordination

At present, Mr. Terry D. Humphrey is an active member of the E&V Team, and is also the Steering Group Subcommittee Chairman for prototype applications development within the JSC Ada Project.

4.10.8 Resolution of Issues

Such issues should initially be addressed within the respective task in which they arise (i.e., the E&V Task or JSC Ada Project). Recommendations should then be developed within that task to resolve such issues. The issues and associated recommendations, should then be presented to the other task leader. Task leaders should work together to obtain resolution. If resolution is unattainable at that level, both task leaders should elevate the issues for

review by STARS' personnel.

4.10.9 Focal Point

The focal point is indicated below :

Jack Garman

Johnson Space Center

Mail Code : FD

Houston, Texas 77058

(713) 483-4788

4.11 Joint Service Software Engineering Environment

4.11.1 Purpose

The Joint Service Software Engineering Environment (JSSEE) is sponsored by the Software Technology for Adaptable Reliable Systems (STARS) Program. A software engineering environment (SEE) is an integrated system that supports mission-critical computer software over the entire life-cycle, from the initial statement of the requirements of the software to the support of the operational software. The purpose of the JSSEE Task Group, which is comprised of selected representatives from the DoD software technical community, is to define and provide a preliminary design of the Joint Service SEE.

4.11.2 Relationship to the E&V Task

One of the goals of the JSSEE Task is to produce a high quality Joint Service SEE. As such, principles which will guide the JSSEE Team include emphasis on production quality tools which reflect human engineering features, and which encourage good software engineering practices. The SEE and E&V Tasks will address common areas of interest and can benefit from one another in the research/technology common to both.

4.11.3 Benefits to the E&V Task

The JSSEE Task will result in the definition and preliminary design of a JSSEE, based upon careful review of life-cycle methodologies, tool functionalities, etc. The rationale which is used by the JSSEE Team to design the JSSEE, will provide useful requirements criteria to be addressed by the E&V Team.

4.11.4 Benefits to the Related Effort/Organization

The evaluation technology developed via the E&V Task will enable the JSSEE Team to assess the tools being incorporated within the JSSEE. The validation technology developed via the E&V Task will enable the JSSEE Team to determine JSSEE compliance with the Common APSE Interface Set (CAIS), which is currently under development by the KIT/KITIA.

4.11.5 Impact on E&V Task Schedules

No schedule impacts are currently identified.

4.11.6 Impact on Related Effort/Organization Schedules

The following E&V Task schedules, with respect to the CAIS Validation Capability (CVC), may impact the JSSEE schedules.

Contract start - 2nd Quarter FY85

Draft CVC - 1st Quarter FY86

Version 1 CVC - 1st Quarter FY86

Version 2 CVC - 4th Quarter FY86

Version 3 CVC - 4th Quarter FY 87

Version 4 CVC - 4th Quarter FY88

4.11.7 Required Level of Coordination

At present, Ms. Ronnie Martin is an active member of both the E&V Team and the JSSEE Team. In addition, E&V Team members John Kramer and Virginia Castor, are on the distribution list for all JSSEE MILNET communications.

4.11.8 Resolution of Issues

Once such issues are identified, they should initially be addressed within the specific task in which the issue arose (i.e., E&V or JSSEE). Recommendations should then be developed within that task to resolve such issues. The issues, and associated recommendations, should then be presented to the other task leader. Task leaders should work together to obtain resolution. If resolution is unattainable at that level, both task leaders should elevate the issues for review by STARS and AJPO personnel.

4.11.9 Focal Point

The focal point is indicated below :

Hank Steubing

Naval Air Development Center

Warminster, Pennsylvania 18974

Commercial : (215) 441-2314

Autovon : 441-2314

4.12 KAPSE Interface Team/KAPSE Interface Team from Industry and Academia

4.12.1 Purpose

The Kernel Ada Programming Support Environment (KAPSE) Interface Team (KIT) is a Navy-led organization sponsored by the Ada Joint Program Office (AJPO). Both the KIT and its companion Industry-Academia Team (KITIA), were formed by a Memorandum of Agreement (MOA) signed by the Assistant Secretary of each of the Services, and the Under Secretary of Defense in early 1982. The KITIA consists of volunteer representatives from industry and universities who provide technical expertise to the KIT. Their purpose is to contribute to the achievement of interoperability of applications databases and transportability of software development tools ("I&T"). In order to accomplish this objective, the KIT/KITIA is defining a Common APSE Interface Set (CAIS) to which all Ada-related tools can be written, thus assuring the ability to share tools and databases between conforming APSEs.

4.12.2 Relationship to the E&V Task

The CAIS will become a MIL-STD in 1985 with revisions anticipated in the following years. As such, a CAIS Validation Capability (CVC) must be developed to enable determination of conformance to the MIL-STD by APSEs which implement the CAIS. One of the goals of the E&V Task is to develop the CVC.

4.12.3 Benefits to the E&V Task

In addition to the definition of the CAIS, the KIT/KITIA activities of developing requirements and criteria, improving upon the STONEMAN definition of an APSE, providing APSE-related terminology and definitions, examining the issue of determining compliance of the CAIS to the original requirements, etc., will provide useful inputs and obviate the need for repetitive activities by the E&V Task. In addition, E&V Tools developed in the future should be portable across CAIS implementations.

4.12.4 Benefits to the Related Effort/Organization

The process of developing a CAIS Validation Capability in parallel with the definition of the CAIS will provide to the KIT/KITIA information related to problems encountered by the E&V Task in understanding CAIS semantics, such as ambiguities and inconsistencies, thus enabling the KIT/KITIA to modify the CAIS definition accordingly. The E&V Task will also help guide the KIT/KITIA in their choice of how to express the semantics and the specifications themselves, based upon experience with what can be validated best.

4.12.5 Impact on E&V Task Schedules

The following KIT/KITIA schedules may potentially impact the E&V Task schedules.

CAIS MIL-STD Version 1 - January 1985

CAIS Draft Version 2 - January 1986

CAIS MIL-STD Version 2 - January 1987

4.12.6 Impact on Related Effort/Organization Schedules

The following E&V Task schedules may impact the schedules of the KIT/KITIA.

CVC contract start - 2nd Quarter of FY85

CVC Version 1 - 1st Quarter of FY86

CVC Version 2 - 4th Quarter of FY86

CVC Version 3 - 4th Quarter of FY87

CVC Version 4 - 4th Quarter of FY88

4.12.7 Required Level of Coordination

There are several E&V Team members who are also members of the KIT/KITIA. Areas of common interest are coordinated through these common representatives. One of these common representatives, John Kramer, is a member of the KIT/KITIA CAISWG as well as a member of the E&V Team CAISWG, and issues specific to CAIS will be coordinated through him.

4.12.8 Resolution of Issues

Issues of concern should be coordinated through the common E&V-KIT/KITIA representatives and raised to the level of Team Leaders if necessary for resolution.

4.12.9 Focal Point

The focal point for the KIT/KITIA is indicated below :

Patricia Oberndorf

Naval Ocean Systems Center (NOSC)

Code 8322

San Diego, California 92152

Commercial (619) 225-6682

Autovon 933-6682

4.13 Methodology Coordinating Team

4.13.1 Purpose

The purpose of the AJPO-sponsored Methodology Coordinating Team is to expand the scope and depth of the Methodman I document produced by Professors Peter Freeman and Anthony Wasserman. It is intended to define the basis for further extension of this work, and provide a preliminary technology for selecting a life-cycle methodology for the preparation and modification of Ada-based software systems.

4.13.2 Relationship to the E&V Task

As technology is developed to evaluate life-cycle software methodologies, it should indicate which methodologies are better than others, and encourage tool set developers to implement those methodologies. At some point, the evaluation of the APSE should determine what methodologies are supported, evaluate them, and indicate to what extent the tool set supports them.

4.13.3 Benefits to the E&V Task

The E&V Task will benefit from the definition of technology to evaluate methodologies implemented on an APSE. Measures could possibly be developed which could be used on E&V tools and tool sets.

4.13.4 Benefits to the Related Effort/Organization

Even though a methodology is evaluated abstractly as good, bad, or somewhere in between, the tool set implementing that methodology can severely impact its usefulness. The E&V technology to evaluate tools and tool sets, should help in determining this aspect of the problem. Also, the E&V technology should assist in characterizing, evaluating, and selecting methodologies, and provide measures to accomplish this.

4.13.5 Impact on E&V Task Schedules

No schedule impacts are currently identified.

4.13.6 Impact on Related Effort/Organization Schedules

No schedule impacts are currently identified.

4.13.7 Required Level of Coordination

At this time, John Kramer is an observer member of the Methodology Coordinating Team, and a reviewer of their plans and documents. At a later time, as the E&V Team begins to develop evaluation criteria concerning the aspects of life-cycle tool sets, at least one joint member would be appropriate. Periodic briefings by each team to the other would be useful.

4.13.8 Resolution of Issues

Issues should be identified, properly specified, and discussed within the team in which the issue originated. The issue should then be forwarded via the MILNET to the chairperson of the other team. The two chairpersons should determine how to address and resolve the issue.

4.13.9 Focal Point

The focal point is indicated below :

Peter Fonash

Department of the Army

DRCDE-SB

5001 Eisenhower Avenue

Alexandria, Virginia 22333

Commercial : (703) 274-9314

4.14 Prototype Advanced Ada Programming Support Environment

4.14.1 Purpose

The purpose of the Prototype Advanced Ada Programming Support Environment (PA-APSE) project is to combine research into advanced APSE features, with support for the KAPSE Interface Team (KIT).

4.14.2 Relationship to the E&V Task

The E&V Task is concerned with developing criteria for judging the quality and value of APSEs. The PA-APSE project investigates potential APSE features which may be found to be required or desirable, and so included in the features considered by E&V.

4.14.3 Benefits to the E&V Task

The PA-APSE project will identify APSE features of potential interest to the E&V Task, and the qualities of those features which are desirable.

4.14.4 Benefits to the Related Effort/Organization

The E&V Task may identify potential APSE features which should be further investigated by the PA-APSE project.

4.14.5 Impact on E&V Task Schedules

No schedule impacts are currently identified.

4.14.6 Impact on Related Effort/Organization Schedules

No schedule impacts are currently identified.

4.14.7 Required Level of Coordination

Mutual information flow is recommended. This can be provided via the communication paths already established between the KIT and the E&V Team.

4.14.8 Resolution of Issues

Such issues should be brought to the attention of the E&V Chairperson, and the PA-APSE contract monitor.

4.14.9 Focal Point

The focal points are indicated below :

Frank Belz (Project Manager at TRW)

TRW DSG

28 August 1984

One Space Park

R2/1127

Redondo Beach, California 92078

Commercial : (213) 535-1623

Patricia Oberndorf (Contract Monitor)

Code 8322

NOSC

San Diego, California 92152

Commercial : (619) 225-6682/7401

4.15 Software Engineering Automation for Tactical Embedded Computer Systems

4.15.1 Purpose

The Software Engineering Automation for Tactical Embedded Computer Systems (SEATECS) project is an internal Naval Ocean Systems Center (NOSC) effort which conducts research into environment construction issues. A set of Top Level Requirements has been developed, and a proposed environment architecture will soon be published. The project also includes an experimental environment which is used to conduct investigations into various proposed environment features. Although SEATECS is not exclusively concerned with APSEs, all of the SEATECS work is applicable to APSEs.

4.15.2 Relationship to the E&V Task

SEATECS is involved in establishing and investigating potential environment features. Such features could be of interest to the E&V Task. In addition, SEATECS seeks to resolve various issues in environment construction which could be of importance to the E&V Task.

4.15.3 Benefits to the E&V Task

SEATECS will identify various aspects of environments which are important to a potential user, but which are often over-looked in current approaches to environment construction.

4.15.4 Benefits to the Related Effort/Organization

The E&V Task may identify issues which are appropriate for investigation using the SEATECS approach.

4.15.5 Impact on E&V Task Schedules

No schedule impacts are currently identified.

4.15.6 Impact on Related Effort/Organization Schedules

No schedule impacts are currently identified.

4.15.7 Required Level of Coordination

Mutual information flow is recommended. This can be provided via the communication paths already established between NOSC and the E&V Team.

4.15.8 Resolution of Issues

Such issues should be brought to the attention of the E&V Chairperson, and the SEATECS project manager, Howard Harvey. The KIT Chairperson, Patricia Oberndorf, will act as liaison as required.

4.15.9 Focal Point

The focal point is indicated below :

Howard Harvey

Code 8322

NOSC

San Diego, California 92152

Commercial : (619) 225-6682/7401

4.16 STARS Human Resources And Engineering Task Area

4.16.1 Purpose

This STARS-sponsored task represents one of the six task areas of the Software Technology for Adaptable Reliable Systems (STARS) program. It focuses on enhancing the skill level of personnel involved in the acquisition, development, and support of mission-critical systems. It also focuses on enhancing the usability of these systems, as well as the usability of software environments.

4.16.2 Relationship to the E&V Task

One of the characteristics to be evaluated by the E&V Task is APSE usability. One of the major concerns of the Human Resources and Engineering Task Area is the design of highly usable systems.

4.16.3 Benefits to the E&V Task

The Human Resources and Engineering Task Area is focusing on steps that can be taken to improve the usability of future environments, while the E&V Task is focusing on developing the technology necessary to evaluate current and future environments. As a consequence of the STARS work, a greater understanding of the characteristics leading to the design of highly usable systems should be gained. This should, in turn, support an approach to evaluating usability which is based on an analysis of these characteristics (as compared to an approach which is purely empirical).

4.16.4 Benefits to the Related Effort/Organization

Progress within the Human Resources and Engineering Task Area depends on an increased understanding of the characteristics leading to highly usable systems (including APSEs). It is anticipated that the E&V Task will lead to the collection of empirical data that will be useful in identifying those characteristics.

4.16.5 Impact on E&V Task Schedules

No schedule impacts are currently identified.

4.16.6 Impact on Related Effort/Organization Schedules

No schedule impacts are currently identified.

4.16.7 Required Level of Coordination

Dr. Elizabeth Bailey is currently serving as a technical consultant to the E&V Team, as well as to the STARS program. She was responsible for drafting the March 1983 technical plan for the Human Engineering Task Area (which has since been merged with the Human Resources Task Area).

4.16.8 Resolution of Issues

Issues should be addressed via coordination between the STARS Joint Program Office and the E&V Team.

4.16.9 Focal Point

The focal point is indicated below :

Carol Morgan

U.S. Navy Acting Program Manager

STARS Joint Program Office

400 Army Navy Drive

Arlington, Virginia 22202

Commercial : (703) 694-0210

4.17 STARS Measurement Task Area

4.17.1 Purpose

This task, which is sponsored by the STARS program, is concerned with the development and use of measures to support evaluations and comparisons of software products, and of the processes associated with software development and support. The strategy for the Measurement Area includes establishing success criteria for the other task areas, performing cost/benefit analyses of various opportunities, collecting baseline data against which to measure progress, instrumenting automated support environments for data collection, and developing techniques for testing hypotheses and models related to software development and

in-service support. Thus, this area is important not only for improving DoD programs, but also for assessing how well the STARS program is meeting its objectives.

4.17.2 Relationship to the E&V Task

The development of quantitative indicies to support comparision is key to both efforts. The Measurement Task Area of STARS is concerned with a broader area than the E&V Task.

4.17.3 Benefits to the E&V Task

One concern of the Measurement Task Area is the instrumentation of automated environments for data collection. Progress in this area will directly benefit the E&V Task.

4.17.4 Benefits to the Related Effort/Organization

The E&V Task is confronted with many of the same issues stemming from the effort to measure APSE characteristics of importance to potential users. Success in the E&V Task will require the development of objective, reliable procedures for measuring these characteristics, some of which (e.g., "usability") are difficult to pin down precisely. Many of the lessons learned, and measurement procedures resulting from the E&V Task will have direct relevance to the Measurement Task Area.

4.17.5 Impact on E&V Task Schedules

No schedule impacts are currently identified.

4.17.6 Impact on Related Effort/Organization Schedules

No schedule impacts are currently identified.

4.17.7 Required Level of Coordination

Dr. Elizabeth Bailey is currrently serving as a technical consultant to the E&V Team, as well as to the STARS program. She served as a reviewer for the March 1983 technical plan for the Measurement Task Area.

4.17.8 Resolution of Issues

Issues should be addressed via coordination between the STARS Joint Program Office and the E&V Team.

4.17.9 Focal Point

The focal point is indicated below :

Carol Morgan

U.S. Navy Acting Program Manager

STARS Joint Program Office

400 Army Navy Drive

Arlington, Virginia 22202

Commercial : (703) 694-0210

4.18 STEP

4.18.1 Purpose

The Software Test and Evaluation Project (STEP), Phases III and IV, is sponsored by the Director Defense Test and Evaluation (DDT&E) and the STARS program. The purpose of STEP is to develop and implement new DoD guidance and policy for the test and evaluation of computer software for mission-critical applications. Principal subgoals include the stimulation of tool development, the support of policy development, and the identification of research issues and directions in the area of software testing. Principal recommendations from the previous STEP phases are intended to establish a chain of test planning, documentation, and evaluation criteria which starts at the most general planning document (the Test and Evaluation Master Plan, or TEMP) and proceeds through the plans and procedures implemented by the Project Offices, development organizations, and independent test organizations. Phases III and IV of STEP, which are currently underway, are designed to define the technology and provide implementation support for these recommendations.

4.18.2 Relationship to the E&V Task

There are at least three areas in which STEP and the E&V Task are related : a) STEP is tasked to develop new guidance statements, as needed, for software test and evaluation (T&E), as well as the necessary implementation methods. Work in this area is intended to address the policy-related issues so that the technology receives the support from above that is needed to put it into practice. This would include any modifications to DoDD 5000.3 and attendant Service regulations, etc., which would require TEMPs to report the results of the evaluation and validation of support software; b) STEP is tasked to produce T&E management and operating plans, and demonstration and qualification procedures for the Software Engineering Institute (SEI). The procedures for inclusion of qualified tools in TEMP specifications and lower-level test plans will also be defined. These tasks address the technology-related problems involved in the qualification of software testing tools for DoD use. This is, in many ways, a subset of the work to develop the E&V technology. In addition, these tasks include the identification of organizations responsible for the application of the technology (e.g., an Independent Evaluation and Validation (IE&V) organization); and c) STEP is tasked to provide functional requirements for APSE test environments. The requirements produced will need to be supported by the E&V technology.

4.18.3 Benefits to the E&V Task

The E&V Task will benefit from STEP's efforts in at least two ways. First of all, the E&V technology will receive the policy support from above which will accelerate its use. Secondly, efforts to develop E&V technology for application to testing tools should benefit from the qualification procedures developed by STEP.

4.18.4 Benefits to the Related Effort/Organization

The efforts of the E&V Task will allow the insertion of demonstrated risk reduction technology into the acquisition cycle. The qualification procedures developed by STEP will be elaborated and inserted into an environment where the standard operating procedures include the evaluation and validation of support software. Furthermore, the functional requirements for APSE test environments to be developed by STEP will be supported by a technology which ensures their implementation.

4.18.5 Impact on E&V Task Schedules

No schedule impacts are currently identified.

4.18.6 Impact on Related Effort/Organization Schedules

No schedule impacts are currently identified.

4.18.7 Required Level of Coordination

At present, R. J. Martin is serving as STEP's liaison to the E&V Task. However, if both tasks are to capitalize upon the obvious opportunities for mutual benefit, additional mechanisms for increased coordination and support should be explored.

4.18.8 Resolution of Issues

Once such issues are identified, they should initially be addressed within the specific task in which the issues arose (i.e., STEP or E&V). Recommendations should then be developed within that task to resolve such issues. The issues, and associated recommendations, should then be presented to the other task. Task leaders should work together to obtain resolution. If resolution is unattainable at that level, both task leaders should elevate the issues for review by DDT&E, STARS, and AJPO personnel, as appropriate.

4.18.9 Focal Point

The DDT&E and STARS focal points, respectively, are indicated below :

Charles K. Watt

Acting Director, Defense Test and Evaluation

Room 3E (1060)

The Pentagon

Washington D.C. 20301

Commercial : (202) 695-7171

Dr Robert Mathis

Director, STARS Joint Program Office

Room 3D 139

400 Army Navy Drive

The Pentagon

Washington D.C. 20310

Autovon : 224-0209

Commercial : (703) 694-0209

4.19 Tactical Ada Guidance

4.19.1 Purpose

The purpose of the Tactical Ada Guidance (TAG) program, which is sponsored by the Air Force Armament Laboratory (AFATL) and the Air Force Computer Resource Management Technology Program Element (64740F), is to demonstrate the use of Ada in a real-time armament system. Specifically, the software in the Medium Range Air-to-Surface Missile (MRASM) Test Instrumentation Controller (TIC) computer is being redesigned and implemented in Ada.

4.19.2 Relationship to the E&V Task

One of the by-products of this effort is the identification of Ada compiler implementation-dependent features that are particularly desirable for armament applications. These features may be useful in defining application-specific metrics for Ada compilers.

4.19.3 Benefits to the E&V Task

The TAG program will result in recommendations for application-specific (i.e., armament) evaluation criteria for Ada compilers.

4.19.4 Benefits to the Related Effort/Organization

No benefits are identified. The TAG program will terminate in November, 1984, prior to any E&V technology transition.

4.19.5 Impact on E&V Task Schedules

No schedule impacts are currently identified.

4.19.6 Impact on Related Effort/Organization Schedules

No schedule impacts are currently identified.

4.19.7 Required Level of Coordination

At present, Chris Anderson is an active member of the E&V Team, and the TAG Program Manager.

4.19.8 Resolution of Issues

Issues identified which relate to the E&V Task will be handled within the E&V Task. Issues identified which relate to the TAG program will be resolved by the AFATL TAG Program Manager.

4.19.9 Focal Point

The focal point is indicated below :

Chris Anderson

Air Force Armament Laboratory/DLMM

Eglin Air Force Base, Florida 32542

Commercial : (904) 882-2961

Autovon : 872-2961

4.20 Telesoft-Ada Programming Support Environment

4.20.1 Purpose

The purpose of this effort is to design and develop an Ada compiler and target code generators, to be hosted on, and targeted for, a variety of computers (e.g., IBM 370, VAX, Motorola 68000, and IBM PC).

4.20.2 Relationship to the E&V Task

The products of these efforts may eventually be evaluated and validated using the requirements and criteria developed by the E&V Task.

4.20.3 Benefits to the E&V Task

The Telesoft-Ada products will be analyzed and may provide useful information for determining the requirements and criteria for evaluating and validating future APSEs.

4.20.4 Benefits to the Related Effort/Organization

The technology developed under the E&V Task may assist Telesoft or its users in identifying APSE features and tools to be incorporated in future Telesoft-Ada revisions and releases.

4.20.5 Impact on E&V Task Schedules

No schedule impacts are currently identified.

4.20.6 Impact on Related Effort/Organization Schedules

No schedule impacts are currently identified.

4.20.7 Required Level of Coordination

Presently, Mr. Terry Humphrey is an active member of the E&V Team and a technical evaluator of the Telesoft-Ada effort for the E&V Team and the NASA Johnson Space Center.

4.20.8 Resolution of Issues

Currently, an issue should be addressed within the task in which it arose. Additional methods for issue resolution are to be determined.

4.20.9 Focal Point

High-level task coordination is to be determined.

4.21 Very High Speed Integrated Circuits

4.21.1 Purpose

The Very High Speed Integrated Circuits (VHSIC) program was initiated in 1975 to meet the present and future DoD needs for complex high speed electronic systems and subsystems.

4.21.2 Relationship to the E&V Task

One of the major components of the VHSIC program is the Integrated Design Automation System (IDAS). IDAS is ultimately to provide an integrated system for the design of VHSIC chips, composed of a set of computer-aided design tools spanning the entire design hierarchy, from system requirement to layout and routing, using a common data base to ensure design integrity. Parts of the CAIS are being proposed as the foundation for IDAS. In addition, the tools developed will become another set of tools in an APSE which the E&V Team must consider evaluating.

4.21.3 Benefits to the E&V Task

The E&V Team can observe the IDAS evolution to better understand the role IDAS-like tool sets will play in future environments, and what is important to evaluate.

4.21.4 Benefits to the Related Effort/Organization

The E&V technology should be useful to the VHSIC program management when it comes time to evaluate the various tool sets and individual tools being proposed.

4.21.5 Impact on E&V Task Schedules

No schedule impacts are currently identified.

4.21.6 Impact on Related Effort/Organization Schedules

No schedule impacts are currently identified.

4.21.7 Required Level of Coordination

The E&V Team should attempt to obtain periodic briefings from the VHSIC JPMO on their plans and schedule. The E&V Team Chairperson should attempt to keep the VHSIC JPMO informed on the status of the E&V Task. IDA will attempt to identify areas of common interest as part of their support to both programs.

4.21.8 Resolution of Issues

Issues should initially be addressed within the respective task (VHSIC or E&V). Recommendations should be developed within that task on how to resolve the issue, and should be forwarded to the other task for consideration. The appropriate Air Force or AJPO/STARS chain should be used for issues which cannot be resolved at the program level.

4.21.9 Focal Point

The focal point is indicated below :

Egbert D. Maynard

400 Army Navy Drive

Arlington, Virginia

Commercial : (202) 697-9216

4.22 Virginia Polytechnic Institute APSE Validation Effort

4.22.1 Purpose

The Virginia Polytechnic Institute (VPI) and State University APSE Validation Effort is a research project conducted by the VPI Department of Computer Science for the AJPO through the Office of Naval Research. The purpose of the effort is to identify and address research issues related to, and supporting, APSE validation. Based on a position paper and proposal delivered to the KITIA meeting in June of 1982 by Tim Lindquist, an initial effort addressing validation needs in an APSE was conducted during the summer of 1982. This study raised issues indicating the need for an APSE model able to accomodate distributed and secure APSEs. It further indicated a need to address validation of a kernel set of APSE facilities to achieve transportability of APSE tools. Subsequent efforts on this project have detailed an APSE model based on the Open Systems Interconnection (OSI) model, and have developed an Abstract Machine approach to specifying the Common APSE Interface Set (CAIS) and a technique for developing a validation suite from the specifications. The project is in the

process of developing specifications for the CAIS Node Model and Process Management sections.

4.22.2 Relationship to the E&V Task

The KIT/KITIA-designed CAIS will become a MIL-STD in 1985. Further, a CAIS Validation Capability (CVC) will be developed through the E&V Task to determine conformance to the CAIS. The specification and validation techniques developed by the VPI APSE Validation Effort relate to both of these activities.

4.22.3 Benefits to the E&V Task

The VPI APSE Validation project specifications for the CAIS Node Model and Process Management sections, will serve as inputs to the development of a CVC. Whether the specifications generated are used for the CAIS, they isolate issues that must be addressed by the CVC. The Abstract Machine descriptions and the technique for generating test cases from the Abstract descriptions, can be used to identify areas the CVC must address.

4.22.4 Benefits to the Related Effort/Organization

This project uses the E&V Team and the KIT/KITIA for review and feedback on its results.

4.22.5 Impact on E&V Task Schedules

The following VPI APSE Validation Effort schedules are of interest to the E&V Task :

September 1, 1984 -- Preliminary Abstract Description of CAIS Node Model

November 1, 1984 -- Preliminary Abstract Description of CAIS Process Management

4.22.6 Impact on Related Effort/Organization Schedules

The E&V Task schedules regarding the CVC impact this effort.

4.22.7 Required Level of Coordination

The Principal Investigator of the VPI APSE Validation Effort (Tim Lindquist), is a technical consultant to the E&V Task, and the E&V Team Chairperson (Virginia Castor), is the Project Monitor.

4.22.8 Resolution of Issues

The focal point for coordination will assist in resolving any issues that arise which may adversely affect either effort.

4.22.9 Focal Point

The focal point is indicated below :

Dr. Timothy E. Lindquist

Department of Computer Science

Virginia Polytechnic Institute and State University

Blacksburg, Virginia 24061

Commercial : (703) 961-7537 (961-6931 messages)

MILNET : LINDQUIST%VPI@RAND-RELAY

4.23 WWMCCS Information System

4.23.1 Purpose

The World Wide Military Command and Control System (WWMCCS) Information System (WIS) effort is a Joint Service Program with the goal of modernizing the existing WWMCCS automatic data processing (ADP), and to upgrade that system with new capabilities to satisfy developing requirements. The WIS program intends to use an Ada designed and implemented software first approach to the upgrade in order not to become locked too early to specific hardware.

4.23.2 Relationship to the E&V Task

WIS is generating a very broad systems development and maintenance environment (Conan). This is to be active on multiple machines over a network. It will have control of code, data, and documentation at several applications levels. It is not a simple APSE, but may contain APSEs. The environment will be developed in Ada, starting from an existing program control and support system. A number of machine-independent tools are being assembled to provide an initial programming capability. Compilers will be evaluated and used as appropriate.

4.23.3 Benefits to the E&V Task

The WIS program will need to evaluate early in the program various tools and tool sets, as well as the tools they develop themselves. This will provide two opportunities for the E&V Team. First, the E&V Task should be able to adopt or learn from any technology for specific evaluations that WIS develops. Second, WIS can possibly be an early user of any E&V technology developed. Present WIS work includes compiler criteria and benchmarks to measure performance against those criteria.

4.23.4 Benefits to the Related Effort/Organization

The WIS program should be able to take advantage of any E&V technology that is developed.

4.23.5 *Impact on* E&V Task Schedules

The following WIS schedules may impact the E&V Task schedules.

Jun 84 - First cut compiler criteria and benchmarks

Jul 84 - Interim initial environment up on IBM 3083

Jul 84 - First draft of final environment (Conan) specification

Sep 84 - First WIS-specific benchmarks

Oct 84 - Block 1 tools operational

Nov 84 - Refined compiler criteria distributed

Nov 84 - Draft Conan environment specification distributed

Apr 85 - Initial environment largely Ada - ported to CUS processor

Apr 85 - Refined Conan environment specification distributed

Jan 86 - Internal delivery of Conan environment

4.23.6 Impact on Related Effort/Organization Schedules

No schedule impacts are currently identified.

4.23.7 Required Level of Coordination

The E&V Team should keep current electronic status from the WIS Joint Program Management Office (JPMO) on their plans and schedules. The E&V Team Chairperson should attempt to keep the WIS JPMO informed on the status of the E&V Task.

4.23.8 Resolution of Issues

Issues shall be addressed within the respective tasks (WIS or E&V). Recommendations should be developed within that task on how to resolve the issue, and should be forwarded to the other team for consideration.

4.23.9 Focal Point

The focal point is indicated below :

Col William A. Whitaker

WIS JPMO/TE

Washington DC 20330

Commercial : (703) 285-5065

Autovon : 365-5065

MILNET : WWHITAKER@ECLB

I. Appendix A

I.1 Acronyms

ACVC	Ada Compiler Validation Capability
ADP	Automatic Data Processing
AFATL	Air Force Armament Laboratory
AFB	Air Force Base
AIE	Ada Integrated Environment
AJPO	Ada Joint Program Office
ALS	Ada Language System
ANSI	American National Standards Institute
APSE	Ada Programming Support Environment
ATVS	Ada Test and Verification System
AVO	Ada Validation Organization
CAIS	Common APSE Interface Set
CAMP	Common Ada Missile Packages
C3I	Command Control Communication and Intelligence
CVC	CAIS Validation Capability
DDT&E	Director Defense Test and Evaluation
DoD	Department of Defense
DoDD	Department of Defense Directive
E&V	Evaluation and Validation
IDA	Institute for Defense Analyses

IDAS	Integrated Design Automation System
IE&V	Independent Evaluation and Validation
ISA	Instruction Set Architecture
I&T	Interoperability and Transportability
JPMO	Joint Program Management Office
JSC	Johnson Space Center
JSSEE	Joint Service Software Engineering Environment
KAPSE	Kernel Ada Programming Support Environment
KIT	KAPSE Interface Team
KITIA	KAPSE Interface Team from Industry and Academia
MAPSE	Minimal Ada Programming Support Environment
MCF	Military Computer Family
MIL	Military
MOA	Memorandum of Agreement
MRASM	Medium Range Air-to-Surface Missile
NASA	National Aeronautics and Space Administration
NOSC	Naval Ocean Systems Center
OSI	Open Systems Interconnection
PA-APSE	Prototype Advanced Ada Programming Support Environment
PE	Project Element
PPG	Program Planning Group

RADC	Rome Air Development Center
SEATECS	Software Engineering Automation for Tactical Embedded Computer Systems
SEE	Software Engineering Environment
SEI	Software Engineering Institute
STARS	Software Technology for Adaptable Reliable Systems
STD	Standard
STEP	Software Test and Evaluation Project
TAG	Tactical Ada Guidance
TCSD	Technical Coordination Strategy Document
TECWG	Technical Coordination Working Group
T&E	Test and Evaluation
TEMP	Test and Evaluation Master Plan
TIC	Test Instrumentation Controller
USDRE	Under Secretary of Defense for Research and Engineering
VAX	Virtual Address Extension
VHSIC	Very High Speed Integrated Circuits
VMS	Virtual Memory System
VPI	Virginia Polytechnic Institute
WIS	WWMCCS Information System
WWMCCS	World Wide Military Command and Control System

II. Appendix B

II.1 TECWG Members

James S. Williamson, Chairperson
Air Force Wright Aeronautical Laboratories

Kevin Chadwick
Canadian National Defense Headquarters

Mark Mears
Air Force Wright Aeronautical Laboratories

Capt. John Taylor
Air Force Logistics Command

EVALUATION and VALIDATION

RELATED TECHNICAL EFFORTS MATRIX

(RTEM)

The following represents the Evaluation and Validation (E&V) Related Technical Efforts Matrix (RTEM), indicating members of the E&V Team, along with potentially related technical efforts/organizations with which each indicated member is involved. This matrix will reside in <EV-INFORMATION> and be continually updated as appropriate by the Technical Coordination Working Group (TECWG).

RELATED TECHNICAL EFFORTS MATRIX

(RTEM)

E & V TEAM MEMBER	1 Ada C3I TEST AND EVAL	2 Ada INTEGRATED ENVRMNT	3 Ada JOINT PROGRAM OFF	4 Ada LANGUAGE SYSTEM	5 Ada TEST AND VER SYS	6 Ada VALIDATION ORG	7 Ada 1750A COMPILER RT	8 AIR FORCE COMP RES MAN	9 CAMP	10 JSC Ada PROJECT	11 JSSE	12 KIT/KITIA	13 METHODOLOGY COORD TEAM	14 PA APSE	15 SEATECS	16 STARS/HUMAN RES & ENG	17 STARS/MEASUREMENT TASK
ANDERSON, C *								X	X								
BAILEY, B																X	X
CASTOR, J												X					
ESTES, N							X										
HUMPHREY, T *										X							
KEAN, L	X	X			X							X					
KNAPPER, R						X											
KRAMER, J *												X	X				
LINDQUIST, T *												X					
MARTIN, R *											X						
OBERNDORF, P												X		X	X		
SCHAAR, B LCDR			X														
TAYLOR, G												X					
WILLIAMSON, J				X													

* SEE ALSO RTEM (CONTINUED)

RTEM (CONTINUED)

```
EEEEEEEE
EE
EEEE
EE
EEEEEEEE

    &

VV      VV
 VV    VV
  VV VV
   VVV
    V
```

	<> RELATED TECHNICAL EFFORT/ORGANIZATION <>																
TEAM MEMBER	18 STEP	19 TACTICAL Ada GUIDANCE	20 TELESOFT APSE	21 VHSIC	22 VPI APSE VALIDATION EF	23 WIS	24	25	26	27	28	29	30	31	32	33	34
ANDERSON, C		X															
HUMPHREY, T			X														
KRAMER, J				X		X											
LINDQUIST, T					X												
MARTIN, R	X																

28 August 1984

The following provides an appendix of acronyms and abbreviations as used in the above Related Technical Efforts Matrix (RTEM) :

APPENDIX

RTEM ACRONYMS/ABBREVIATIONS

AJPO	Ada Joint Program Office
APSE	Ada Programming Support Environment
CAMP	Common Ada Missile Packages
COMP	Computer
COORD	Coordinating
C3I	Command Control Communication and Intelligence
EF	Efforts
ENG	Engineering
ENVRMNT	Environment
EVAL	Evaluation
JSC	Johnson Space Center
JSSEE	Joint Service Software Engineering Environment
KIT	KAPSE Interface Team
KITIA	KAPSE Interface Team from Industry and Academia
MAN	Management
OFF	Office
ORG	Organization
PA	Prototype Advanced
RES	Resource
RT	Runtime
SEATECS	Software Engineering Automation for Tactical Embedded Computer Systems
STARS	Software Technology for Adaptable Reliable Systems
STEP	Software Test and Evaluation Project
SYS	System
VER	Verification
VHSIC	Very High Speed Integrated Circuits
VPI	Virginia Polytechnic Institute
WIS	WWMCCS Information System (Note : "WWMCCS" stands for Worldwide Military Command and Control System.)

APPENDIX E

Evaluation and Validation

Public Coordination Strategy Document

Version 1.0

6 March 1984

Table of Contents

1. INTRODUCTION

1.1 Objective

The purpose of the Evaluation and Validation (E&V) team is to develop the techniques and tools which will provide a capability to perform assessment of Ada Programming Support Environments (APSEs) and to determine conformance of APSEs to the Common APSE Interface Set (CAIS). As the E&V technology is developed, it will be made available to the community for use by DOD components, industry, and academia as deemed appropriate by the respective organization. The objective of the E&V Public Coordination Working Group (PUBWG) is to transition this technology to the public as it becomes available. This document describes the strategy for accomplishing this transition.

1.2 Background

Currently there is little data available on correlating specific APSE capabilities with project requirements. As stated in section 1.1, one of the goals of the E&V task is to provide an evaluation capability for those features of APSE components for which there exists no validation capability or formal standard (e.g., Ada compiler's implementation dependent features, run-time system characteristics, etc.). Since this information is critical to the success of the software development process, it is essential that emerging evaluation techniques resulting from the E&V task be provided to the public as soon as they become available.

Similarly, in order to ascertain whether data & tools from one APSE will be able to be transported to another APSE, some metrics must be applied to the APSE interface (i.e., the CAIS). These metrics will be developed by the E&V task in order to provide a CAIS validation capability. Again, this information is vital to software program managers and designers who plan to achieve maximum software portability and should be made available to the public as it emerges. It is the intent of this document to describe the procedures to facilitate the transition of the E&V technology to the public.

2.0 SCOPE

In order to accomplish the objectives described in section 1.1, it is essential that the E&V team maintain open channels of communication to the public. The public coordination strategy outlined in this document primarily focuses on communication originating from the E&V team to the public and associated feedback from this communication. Technical information related to E&V orginating outside the Team will be monitored by the E&V

Technical Working Group (TECWG) and transmitted to the appropriate E&V working groups. The mechanisms supporting the outflow of information from the Team to the public are described in section 3 and appendices A-G.

3. APPROACH

Several mechanisms for communication to the public have been identified to assist team members in the public exchange process. These mechanisms will be outlined in the following subsections.

3.1 Briefings

It may be appropriate for team members to occasionly present briefings to the public. A current set of "official" E&V vugraphs (see Appendix E) will be maintained by the PUBWG for use by any team member. It will not be necessary to get permission from the E&V Chairperson to brief. However, following a presentation, submission of the "Public Exchange Record" to the Chairperson, with a copy to the Team is required. The format for this form is listed in Appendix D and on MILNET file <   >.

Briefings may be presented to DoD organizations, committees and conferences as well as to industry and academia.

A list of candidate organizations is presented in Appendix A, along with the primary mission and point of contact.

An E&V update will be presented by the Chairperson or designated alternate routinely at the AdaJUG Government Corner and the Ada TEC Environment/Standards subcommittee session.

3.2 Papers

Various team members may wish to submit papers to professional journals. The paper must be submitted to the E&V Chairperson for review and approval prior to publication. A copy of the approved paper should be forwarded to the PUBWG for inclusion in the Annual Report. A list of candidate publications is given in Appendix B, along with procedures for document submittal.

Also the author of the paper should complete the PUBWG form listed in Appendix D for inclusion in the E&V Project Reference List maintained by the PUBWG on the ARPANET.

3.3 LCF Newsletter

A quarterly report based on the official minutes from the E&V meetings will be submitted to the LCF Newsletter on a routine basis. This report will be prepared by the PUBWG and submitted to the E&V Chairperson for approval prior to publication.

3.4 E&V Information

Another mechanism for communication with the public is via the ARPANET E&V Information account. The most recent E&V minutes will be presented in "E&V Today" and past minutes will be filed with file reference by date given in "E&V Yesterday". These minutes will be taken by the PUBWG and submitted for Team comment and approval prior to entry into the E&V Information account. A hardcopy will also be incorporated in the Annual Public Report. The format for the minutes is presented in Appendix F.

3.5 E&V Quarterly Report

As stated in section 3.3, the PUBWG will prepare a brief E&V quarterly report based on the E&V minutes. After being reviewed by the Team and approved by the E&V Chairperson, the report will be sent to the LCF Newsletter, put on the E&V Information ARPANET account and be available for publication or "handouts" at appropriate conferences.

3.6 Project Reference List

A list of E&V related documents will be kept in the Project Reference List on the E&V Information account. This list will be maintained by the PUBWG. Team members should contribute to the list by filling out the template listed in Appendix D and sending it to the Chairperson with a copy to the team. The list will not only inform the public about various E&V related studies but also keep the Team up to date on any related technology.

3.7 E&V Public Report

An E&V Public Report will be published annually in order to provide the public with information on the activities of the E&V Team. The E&V Public Report will contain the recorded minutes of all E&V Team meetings as well as all position papers prepared by E&V Team members. The E&V Public Report will also contain E&V position papers written by industry/academia participants in the annual E&V workshop, as well as all documentation which results from the E&V workshop. The format of the E&V Public Report is described in Appendix G.

APPENDIX A

Organizations

The following organizations have been identified as possible candidates for E&V related presentations. DoD and industry/academia organizations are listed separately.

DoD Organization and Conferences

AFSC Embedded Computer Resource Focal Point Group

This group consists of representatives from the AFSC laboratories and product divisions associated with embedded computer resources. Meetings are held approximately three times yearly. Attendance is usually limited to members. To present a special briefing of interest contact your AFSC ECR focal point or Col Kenneth Nidiffer, AFSC/ALR, Andrews AFB, DC 20332, AV 858-5731.

AFSC Software Technology Working Group

This group consists of representatives associated with software technology from the AFSC laboratories. Meetings are held approximately four times per year. Only official members may attend. To present a special briefing contact your AFSC representative or Lt Col Jim Riley, AFSC/DLA, Andrews AFB, DC 20334, AV 858-2482.

Armament/Avionics Stadardization Conference

This annual conference (usally Sept) is jointly sponsored by AFSC and ALC. Candidate presentations should conform to panel issues. The chairman of the standardization Panel is Lt Col Frank Grosso, Hq USAF/RDPV, AV 227-7715

KAPSE Interface Team/KAPSE Interface Team Industry & Academia (KIT/KITIA)

The purpose of the KIT and KITIA, under the direction of the AJPO is to develop a standard set of KAPSE interfaces to ensure the transportability of tools and the interoperability of data between conforming APSES. The E&V Team will interact with the KIT/KITIA for information exchange via briefings and intermembership (several members of the E&V Team are also members of the KIT/KITIA. The Chairperson of the KIT is Patricia Oberndorf, NOSC, San Diego, CA.

METHODMAN

The purpose of the METHODMAN effort, under the direction of the AJPO, is to develop requirements and encourage the development of methodologies to support the entire software development life-cycle. One of the goals of the METHODMAN effort will be the construction of a complete set of tools to support a selected methodology. The E&V Team will interface with the METHODMAN effort for information exchange, particularly in the areas of tool definitions, evaluations and validation. The Chairperson is __________________.

AJPO Director's Advisory (ADA) Board

The purpose of the ADA Board is to advise the director of the AJPO with regard to policy and issues related the Ada program. The AJPO Director is Dr. Robert Mathis, AV 224-0208.

Software Technology for Adaptable Reliable System (STARS)

The STARS program, under the direction of the DoD, was established to develop and promote new software technology. The Acting Director of the STARS program office is Col Vance Mall, AV 224-0208.

Ada Validation Organization (AVO)

The AVO, under AJPO direction, has established formal procedures for testing Ada compilers to ensure compliance with the language standard (ANSI/MIL-STD-1815A-1983). The E&V Team will interact with the AVO in the area of validation requirements and procedures. The Deputy Director of the AVO is Robert Knapper, Institute for Defense Analysis.

Other Organizations

AdaTEC

This professional association meets four times yearly. Technical topics associated with the use or implementation of Ada are welcome. For more information concerning the appropriate session to present a briefing, contact Jean Whitaker, Hughes Aircraft Co. (714) 7320-9231.

AdaJUG

This organization brings together representatives from industry, academia and the Government interested in standardizatin and language control activities, compilers and tools; applications and development efforts associated with JOVIAL and Ada. The Government Corner is an appropriate session to brief short presentations (15-20 min) concerning Government sponsored Ada Activities. The Chairman of this session is Ron Vokits, ASD/AXT, Wright-Patterson AFB, OH 45433, AV 785-5941. For longer presentations contact the AdaJUG Chairperson Donna Grant at (916) 920-3663.

National Security Industries Association (NSIA)

Members of this association are defense contractors. Open National Conferences focusing on special topics are held several times a year. For further information contact W. M. McMurray, General Dynamics (314) 851-8910.

Institute of Electrical and Electornics Engineers (IEEE)

This professional organization has over 210,000 engineers and scientist members. There are numerous meetings and special technical conferences held annually. For more information contact Eric Herz, Executive Directorate (212) 705-7900 or write 345 East 47th Street, New York, NY 10017.

Association for Computing Machinery (ACM)

This professional organzation has over 53,000 members associated with computing and data processing. There are over 31 special interest groups. National conferences are held annually (usually in October). For more information contact Sidney Weinstein, Executive Director at (212) 869-7440 or write 11 West 42nd Street, 3rd Floor, New York, NY 10036.

APPENDIX E

Publications

The following publications have been identified as candidates for publishing E&V related paper. Procedures for document submittal are also included.

Computer Magazine (IEEE)

Articles that cover all aspects of computer science are welcome. Articles are usually survey or tutorial in nature. Submit six copies of the manuscript including illustrations, references, and authors' biographies to the editor-in-chief:

Stephen S. Yau
Dept of EE and Computer Science
Northwestern University
Evanston, IL 60201
Telephone: (312) 492-3641

Defense Electronics

Articles covering aspects of computer science that are relevant to the DoD community are welcome. Send to:

EW Communications, Inc.
1170 East Meadow Drive
Palo Alto, CA 94303-4275
Telephone: (415) 494-2800

Ada Letters (ACM)

Information dealing with all aspects of Ada are welcome. "Short Notices" announce meetings or publications. "Letters to the Editor" raise issues or answer them. Articles typically deal with in-depth technical topics related to the use of Ada. "Ada Events" announce significant events of major interest to the Ada community.

Short Notices
Letters to the Editor

Mary S. Van Deusen
34 Archer Street
Wrentham, MA 02093
(617) 384-2526

Articles — Ronald F. Brender
DEC
110 Spit Brook Rd
ZK02 - 3/N30
Nashua, New Hampshire 03062
(603) 881-2088

Ada Events — Robert I. Eachus
Honeywell SSPD
300 Concord Road
Billerica, MA 01821
(617) 671-2907

Submissions should be single-spaced with no page numbers, and may be printed two-up. Submission deadlines Aug 31, Oct 31, Dec 31, Feb 28/29, Apr 30 and Jun 30.

Communications of the ACM

Papers on all aspects of computing are solicited. (For format, see July 1982 issue.) Submit to:

Nicolas Mokhoff
ACM Headquarters
11 West 42nd Street
New York, NY 10036
(212) 869-7440

JOVIAL Language Control Facility Newsletter

Brief articles on announcements related to Ada activities are welcome. Submit to:

ASD/ADOL
Wright-Patterson AFB, OH 45433
(513) 255-4472/4473
AV 785-4472
LCF at WPAFB-JALCF

APPENDIX C

Project Reference List

The following documents have been idenfitied as E&V related. This list is available on the ARPANET (E&V Information account). The list presented in this appendix is an initial one and will expand as other E&V related documents are identified and as team members and workshop participants contribute to the E&V knowledge base. The list is in alphabetical order by author.

APPENDIX D

PUBWG Forms

There are three forms for the submission of data to the Team: the Public Exchange Record, the Project Reference List Submittal Form and the E&V Status Report.

The Public Exhange Record shall be submitted following the presentation of a paper or "official" E&V briefing. The form shall be submitted to the Chairperson with a copy to the Team.

The format of the form is listed below and may also be found in <CASTOR.PUBWG>.

PUBLIC EXCHANGE RECORD*

TYPE OF EXCHANGE: (Briefing, paper etc.)

SPECIFIC TOPIC:

DATE:

PLACE OR PUBLICATION:

ATTENDEES:

PRESENTER:

MATERIAL PRESENTED: (Send via net to Castor with copy to the Team or send by mail to Virginia Castor, AFWAL/AAAF-2, WPAFB, OH 45433 and Chris Anderson, AFATL/DLMM, Eglin AFB, FL 32542.)

FEEDBACK:

*Team members should submit this form to Castor with a copy to the Team.

PROJECT REFERENCE LIST SUBMISSION FORM*

TITLE:

DATE:

AUTHOR(S):

AFFILIATE:

SPONSOR:

ABSTRACT:

RELATIONSHIP TO E&V:

TO ORDER:

*Team members should submit this form to Castor with a copy to the Team.

E&V Status Report

The E&V Status Report shall be submitted by working group chairs to the E&V Chairperson at the quarterly E&V meeting. The format is as follows.

WORKING GROUP:

DATE:

PERSONNEL CHANGES:

DELIVERABLES DUE THIS QUARTER:

ACCOMPLISHMENTS THIS QUARTER:

UNRESOLVED PROBLEMS OR ACTION ITEMS:

PROJECTED WORK FOR NEXT QUARTER:

DELIVERABLES DUE NEXT QUARTER:

APPENDIX E

Vugraphs

The E&V Team maintains an "official" set of vugraphs to support individual Team member presentations at DoD and industry/academia meetings and conferences. The initial set is presented in this appendix. It is envisioned that this set will continuously evolve as the E&V effort matures.

APPENDIX F

E&V Minutes Format

The minutes of E&V quarterly meetings are presented on the ARPANET in the E&V_Information account and in the E&V Public Report. The format of the minutes is outlined below.

Minutes

of the

EVALUATION & VALIDATION (E&V) MEETING

Date

Wright-Patterson Air Force Base, Ohio

1. Date

1.1 Topic 1

1.2 Topic 2

1.N Topic N

2. Date

2.1 Topic 1

2.2 Topic 2

2.N Topic N

Appendix G

E&V Public Report Format

The E&V Public Report will be published annually. It will be available to the public through the National Technical Information Service (NTIS). The format for this report is listed below.

E&V Public Report

I. Introduction

II. Team Proceedings

A. E&V Minutes 7-8 Dec 1983

B. E&V Minutes 7-8 Mar 1984

C. E&V Minutes 6-7 Jun 1984

D. E&V Minutes 5-6 Sep 1984

III. E&V Documentation

A. E&V Effort Planning

B. E&V Plan

C. Working Group Status Reports

D. E&V Team Point Papers

IV. E&V Workshop

A. Minutes

B. Point Papers

C. Attendees

Appendix A E&V Team Members

Appendix F

Minutes

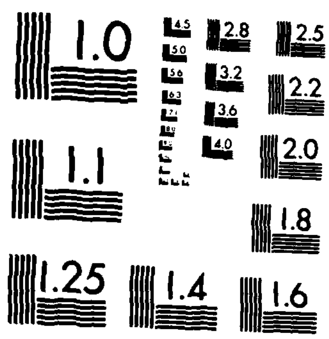
of the

EVALUATION & VALIDATION (E&V) MEETING

7-8 December 1983

Wright-Patterson Air Force Base, Ohio

AD-A153 609 EVALUATION AND VALIDATION (E&V) TEAM PUBLIC REPORT 4/6
VOLUME 1(U) AIR FORCE WRIGHT AERONAUTICAL LABS
WRIGHT-PATTERSON AFB OH V L CASTOR 30 NOV 84
UNCLASSIFIED AFWAL-TR-85-1016-VOL-1 F/G 14/2 NL



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

Table of Contents

1. Wednesday, 7 December 1983

1.1 Welcome and Introductions

The first E&V Team meeting began with a welcome to all by Jinny Castor, followed by an explanation of the registration materials (E&V Meeting Agenda, E&V Plan, E&V Letters, etc). Jinny then acknowledged the AFWAL personnel whose efforts had contributed to setting up the facilities and materials for the meeting: Bonnie Conover, Angela Crumley, Rich Wallace, and Jimmy Williamson. She then introduced the technical consultants who had been contributing to the initial efforts of the E&V Task: Paul Dobbs (General Dynamics), Jack Kramer (Institute for Defense Analyses), and Tim Lindquist (Virginia Polytechnic Institute). Recognition was also given to those individuals who were scheduled to speak during the day, but who had not yet arrived: LCDR Brian Schaar (Ada Joint Program Office), Maj Izzy Caro (Air Force Wright Aeronautical Laboratories), and Bob Knapper (IDA). Each team member then provided a self-introduction and an overview of the activities of his/her respective organization.

1.2 Air Force Perspective of E&V

Major Izzy Caro then discussed what the E&V Task meant to the Avionics Laboratory and to the Avionics community as a whole at Wright-Patterson Air Force Base. He first expressed his philosophy as to why E&V is important. He stated that the E&V technology which is developed will provide to the Air Force and DoD, for the first time, a capability to quantify and qualify software tools to be used in developing weapon systems. He noted the history of JOVIAL compilers which resulted in several compilers now available for selection and use by the Air Force. There is no quantifiable means for selection of such compilers by contractors or the Air Force. He stated that for Ada Programming Support Environments the Air Force will use, not only the Ada Language System (ALS) and Ada Integrated Environment (AIE), but also industry developed Ada Programming Support Environments (APSEs) and it will be essential to develop the E&V technology to enable judicious selection.

Major Caro cautioned the E&V Team against use of a "frontal assault" in the development of the E&V technology. He emphasized the need to develop an initial effective capability in the short term, with incremental developments and enhancements throughout

the E&V Task. He stated that program managers need to make a quantitative assessment of support software tools "now" because software is such a major factor in the cost of weapon systems development.

In closing, Major Caro expressed his personal support, as well as the support of the Avionics Laboratory and the Air Force, for the E&V Task and reemphasized the need for short-term, as well as long-term, results from this effort.

1.3 E&V Task Overview

Primarily for the benefit of the E&V Team members who were unfamiliar with the concept of the E&V Task, Jinny Castor presented an overview of the Task. She stated that the viewgraphs she was using in her presentation were those used to brief the the E&V Task at the Ada tri-service review in Dallas, Texas in October, 1983. She explained that the purpose of the quarterly Ada tri-service reviews is to enable the AJPO and service representatives to discuss the plans and progress of Ada related efforts. She told the team members that information on the E&V Task was presented in considerably more detail in the E&V Plan, which she would brief later in the day. Her high level overview of the E&V Task began with an introduction to the concept of an APSE and the Common APSE Interface Set (CAIS). She then elaborated on the distinction between "evaluation" and "validation" (evaluation representing a qualitative assessment of an APSE component for which no objective standard exists, and validation representing a quantitative assessment of an APSE component for which both a metrics capability and a standard exist). She described the evolution of the E&V Task, which originated as the result of efforts such as the Ada Validation Organization (AVO), the Kernel APSE (KAPSE) Interface Team (KIT)/KAPSE Interface Team from Industry and Academia (KITIA) and its CAIS development, the National Bureau of Standards (NBS) APSE Taxonomy, the AFWAL Evaluation Criteria Document, and the Virginia Polytechnic Institute APSE validation studies. She described the objectives of the E&V Task: identifying APSE components, classifying APSE components for evaluation or validation, acquiring and developing tools/techniques for application of E&V technology, and promoting the use and acceptance of the E&V technology. She then described the relationship of the E&V Task to other Ada related efforts.

Jinny then listed the short term activities (E&V Workshop

planning, education and training for E&V Team members) and long term activities (E&V Requirements Document, E&V Reference Manual, E&V Guidebook, annual E&V Plan, annual E&V Technical Report, Formal Qualification Testing of the DoD APSEs) for the E&V Team. She then listed the significant events during the fourth quarter of fiscal year 83 which lead to Air Force official acceptance as lead service in the E&V task. She also emphasized what the goals were for the first quarter of fiscal year 84 and indicated that they had been accomplished.

In conclusion, Jinny elaborated on the goals of the E&V Task: (1) to provide a capability to determine APSE conformance to the Common APSE Interface Set; (2) to promote the development of quality APSEs; (3) to transition Ada and E&V technology to the organizations represented on the E&V Team; and (4) to transition E&V technology to DoD and public organizations.

1.4 Ada Joint Program Office

LCDR Brian Schaar began his presentation by expressing appreciation for the initialization of the E&V Task and the efforts to be undertaken by the representatives on the E&V Team. He then described the organizational structure of the AJPO and described the roles of the Director and Service Deputy Directors at the AJPO. Brian illustrated the relationship of the AJPO to the Office of the Secretary of Defense. He also provided the organizational structure of the Computer Software Systems (CSS) Directorate and described the roles of the personnel in the CSS.

Brian then outlined the AJPO Tasking Philosophy for the initialization of tasks such as E&V. The first step is to identify a specific need for a task. The second step is to identify a potential leader to provide technical direction to the task. The third step is to develop a formal mechanism for the establishment of the task. He described how those steps were accomplished for the E&V Task and he then listed three primary tasks sponsored by the AJPO: KAPSE Interface Team (Navy); Evaluation & Validation (Air Force); and Methodologies (Army). He emphasized that an effort is made to distribute the workload among the services and stated that the progress of each of these tasks is reported at the quarterly Ada tri-service reviews, at which AJPO personnel and service representatives have the opportunity to discuss and influence the plans and progress of each task.

1.5 Philosophy of Environments

Jack Kramer (IDA) provided a presentation on the philosophy of environments. Because there were team members who were not familiar with the details of the Ada Program, Jack indicated that this presentation also provided information on the history of the Ada Program. Jack began his presentation by stating that in 1975 many problems were identified as relevant to the development of software for military missions. Using several charts for illustration he indicated the escalating costs associated with software development. These charts also illustrated the decrease in the percentage of costs due to hardware and the increase in the percentage of costs due to software. He described how the High Order Language Working Group (HOLWG) was established in 1975 to formulate requirements for a DoD HOL, to evaluate existing language approaches, and to recommend a minimal set of common HOLs. The resulting recommendations of the HOLWG were the development and standardization of a single modern HOL, the use of integrated programming support environments, and the use of modern software development practices. Jack stated that the goal then, and as presently reflected by the Software Technology for Adapable, Reliable Systems (STARS) program now, was to improve productivity while achieving greater system reliability and adaptability.

Jack then listed the characteristics associated with mission critical computer systems (real time constraints, automatic error recovery, concurrent control, non-standard input/output, and large long-lived software programs which require continuous change). He stated that Ada and the APSE will control software cost and improve software quality through facilitation of modern software practices, automation of life-cycle tasks, increased software sharing, and increased personnel portability.

Jack described the history of the Ada language development and identified the many problems which will be addressed by environments (policies, automated tools, procedures, methods, education, etc.) He listed the goals of the Ada Program (implement Ada as a standard, promote the adoption of Ada, provide Ada education and training, and provide support systems) and indicated that mission applications represented only one small portion of the software arena. He described the development of the STONEMAN document and provided a list of tools and components of an APSE. He stated that the cost and quality of software can be controlled through the application of engineering disciplines to the life-cycle process, the automation

of many life-cycle tasks, and the reusability of software and training products.

In closing, Jack stated that the purpose of APSEs is to support the development and maintenance of embedded computer software throughout its life cycle. He emphasized that it is important to provide initial APSE capabilities now (such as the ALS, AIE, ALS/N) which will provide useful foundations on which to build in the future. He cited the CAIS and data base conventions as examples on which to build in order to ensure tool cooperation. He indicated that the E&V Task will also have to provide evaluation and validation capabilities for the data which is captured within an APSE.

1.6 KAPSE Interface Team

Jack Kramer presented an overview of the KAPSE Interface Team (KIT), beginning with the background of the KIT's establishment and its initial objectives. He stated that the KIT was formed as a result of a January 1982 tri-service Memorandum of Agreement which established a Navy led DoD Team chartered to formulate interface standards: (1) to facilitate movement of tools and data between APSEs and (2) toward which the AIE, ALS and all other DoD APSEs could evolve. He described how the KITIA was established for the purpose of augmenting the level of expertise on the KIT and providing an industry and academic orientation to the KIT effort. Jack reviewed the STONEMAN APSE model and indicated that the STONEMAN document was not sufficient to accomplish the goal for which the KIT was established because: (1) STONEMAN assumed portability of tools would be achieved by rehosting a single KAPSE; (2) the DoD had already initiated the development of two different APSEs (and KAPSEs); (3) industry was developing APSEs; and (4) the objective of the Ada Program was to reduce cost by having portability of tools, data bases, and programmers.

Jack described the composition of the KIT, indicating the primary representation by Navy personnel, with additional representation from the Air Force, Army, Canadian National Defence, and the support contractors currently developing the DoD APSEs. He also described the composition of the KITIA and reviewed the process by which KITIA members were solicited and accepted for representation. He stated that the purpose of the KIT was to propose DoD standards to enable the interoperability and transportability of data and tools among APSEs and he also stated that the data base area would have to be addressed by the

E&V effort because of the need to capture data used during the development process for later use during the maintenance phase.

Jack elaborated upon the KIT objectives and accomplishments, indicating the level of effort which has thus far gone into the development of the CAIS and Requirements and Criteria Document. He also stated that the E&V Requirements Document would likely be as complex a task for the E&V Team. Jack mentioned that the Compliance Working Group (COMPWG) of the KIT would provide very useful information into the E&V Task. He concluded his presentation by listing the KIT plans, which include: (1) a completed Requirements and Criteria Document by the end of this winter; (2) a companion guidelines document for use by tool-builders and KAPSE writers; (3) evolution of the CAIS document and its submission for standardization; and (4) completion of all work by 1986.

Jack's presentation drew numerous questions from the E&V Team, primarily with respect to better understanding of the KAPSE concept and the CAIS. While answering these questions Jack emphasized that the E&V effort will provide valuable input to the standardization process. He noted the similarity between the development of the Ada language at the same time the Ada Compiler Validation Capability (ACVC) was being developed and the development of the CAIS at the same time the CAIS Validation Capability (CVC) will be developed. The parallel development process should result in good insight into the semantics of the CAIS through development of its validation capability.

1.7 Ada Compiler Validation

Bob Knapper (IDA) began his presentation on Ada Validation by acknowledging the efforts of Tom Probert (IDA), who had originally been scheduled to give the presentation but who had been unable to attend. Bob first explained that Ada Compiler Validation means determining conformance of the Ada Language Processor to the language specifications of ANSI/MIL-STD-1815A-1983. At the present time, Ada Compiler Validation does not measure usability or fitness and does not guarantee that anomalies will not occur. He indicated that validation of Ada compilers is necessary in order to enhance transportability, reusability and reliability of Ada software, to prevent Ada dialects, and to support the maintenance of the standard. He stated that there are three major components involved in Ada compiler validations: (1) the Ada Validation

Organization (currently consisting of Bob Knapper and Tom Probert of IDA); (2) the Ada Compiler Validation Capability (ACVC) test suite; and (3) the AJPO Director's Advisory (ADA) Board (a multinational organization which serves as the "ultimate court" for resolution of language validation issues).

Bob explained that the purpose of the AVO is multifold: (1) to encourage, measure, and enforce conformance to the Ada language standard as well as other related standards such as the future CAIS standard and software tools; (2) to provide technical assistance to implementers; and (3) to provide research in validation and software quality assurance.

Bob explained that the current ACVC consists of a test suite (1600 programs indexed to the LRM and the Implementer's Guide), the Implementers Guide which identifies semantic ramifications in interaction between language features, and the testing tools (report package and analysis tools). He explained the six test classes and reviewed the release schedule for the ACVC. Version 1.4 (due April, 1984) will contain a minimum of 500 additional test programs. Version 1.5 (due November, 1984) will represent stabilization of the test suite. At the request of Brian Schaar, Bob reviewed the concept of the host-target-operating system "triple" by which compilers are validated. Bob also explained that a Fast Reaction Team has been established to resolve fine points of language issues as they apply for a pending validation, with a turnaround time of 72 hours. He also stated that the ADA Board, which will be implemented as a Federal Advisory Committee, serves in an advisory role to the AJPO in the areas of language standards, validation, environments and liaison with other organizations and that the ADA Board will be closely involved with the CAIS validation activities of E&V.

Bob continued his presentation with a discussion of the basic assumptions which are made for validation: (1) that Ada language processors are designed to implement ANSI/MIL-STD-1815A Ada (and not the validation test suite); (2) that it is in the interest of Ada implementers as well as consumers to implement a quality product; (3) that market pressure is every bit as powerful, perhaps more so, than DoD regulations; and (4) that rapid testing and analysis of results are essential to acceptance of the validation requirement. He stated that similar assumptions will be applicable to CAIS validation.

Bob briefly reviewed the current practices used in validation. Items which he elaborated upon included the fact that on-site

validation procedures are usually accomplished within 2-3 days, with a "pass/fail" verdict obtained by the AVO within an approximate 2 week period. Validation summary reports are available to the public through NTIS. Bob stated that the current 30 day pre-release period of the ACVC official test suite will be extended to 60 days with the release of Version 1.5. He also stated that the validation test suite will be enhanced to provide additional evaluation capabilities. He emphasized that no official validation analysis is done on site and he briefly described the appeals process which is currently used. He suggested that the current validation procedures used for Ada compilers would provide very good pointers to the validation procedures which will be used for the CAIS.

Bob listed the initial problems which had to be overcome by the AVO: (1) implementer animosity towards the AVO; (2) an essentially untried test suite; (3) actually enforcing policies and procedures; and (4) user community pressure. In response to a question on the ROLM compiler validation, Bob indicated that out of the 1600 tests, approximately 400 tests are implementation dependent. Thus one contractor could pass validation without having passed all of the 1600 tests. Bob explained how an Ada Implementation Database has been used to record pertinent information regarding each validation, and he strongly advised the use of a similar database technique for CAIS Implementation validations.

Bob listed future plans of the AVO (60 day ACVC pre-release time, vendors providing after hours access for validation team, licensing of satellite facilities, and incorporation of additional tests into the ACVC test suite). He also indicated that the AVO will participate in E&V of environments, certification of transportability and interoperability, testing for appropriate optimization, and validation of DIANA. For the benefit of the E&V Team he reviewed "lessons learned" by the AVO and listed additional problem areas still to be overcome by the AVO (resolution of the host-target-operating system triple, validation of totally embedded systems, and resolution of multiple error tests).

Following the conclusion of Bob's presentation, numerous questions were asked regarding the host-target-operating system triple implications for embedded systems, which Bob clarified. In answer to a question regarding availability of Ada compilers in the future, Bob estimated that he expected approximately 5 new validations within the next 6-9 months.

1.8 E&V Plan

Jinny Castor provided a presentation on the E&V Plan, dated 30 November 1983. She emphasized that the E&V Plan is considered a "living" document, and will be revised in 1984. However, for the present, the E&V Plan provides the direction for the activities of the E&V Team. Chapter 1 provides an introduction which identifies the objective of the plan and provides background information which led to the development of the E&V Task. Chapter 2 lists 11 specific objectives of the E&V Task. In her review of the objectives, Jinny emphasized that the E&V Team will not be performing the actual evaluations and validations, but rather, will be developing the technology by which such evaluations and validations will be performed. Chapter 3 provides the technical approach to E&V and includes a description of the APSE concept and the E&V Classification Schema. Chapter 4 provides the management approach to E&V, and lists the various organizations involved in the management structure. This chapter also provides descriptions of the various working groups into which the E&V Team is to be partitioned (Requirements Working Group (REQWG), Technical Coordination Working Group (TECWG), APSE Analysts Working Group (APSEWG), CAIS Working Group (CAISWG), and Public Coordination Working Group (PUBWG)).

Chapter 5 of the E&V Plan describes the relationship of the E&V Task to other organizations and activities. Chapter 6 lists and describes all of the deliverables anticipated from the E&V Task. Chapter 7 provides a work breakdown structure. Chapter 8 provides schedules and milestones for the E&V deliverables, meetings, and contractual efforts. Chapter 9 concludes with a list of references used within the document.

Following her "walk through" of the E&V Plan, Jinny requested the E&V Team members to carefully review the responsibilities of the working groups so that, on the next day, each Team member could select the working group in which he/she preferred to participate.

1.9 Open Discussion

During the open discussion period which followed, clarifications were made as to who would participate in the E&V Workshop (selected E&V Team members and industry/academia representatives), who would prepare the minutes of the E&V meetings (Jinny for the first meeting, the PUBWG for all following meetings), and how to resolve possible working group selection distributions (allow open selection initially). The E&V meeting was then adjourned for the day.

2. Thursday, 8 December 1983

2.1 Announcements/E&V ECLB Accounts Info

Jinny Castor opened the second day portion of the E&V meeting with a schedule for all E&V meetings in 1984 (7-8 Mar, 6-7 Jun, 5-6 Sep, and 5-6 Dec), all of which will be held at Wright-Patterson Air Force Base. She then explained that the primary means of communication among E&V Team members will be via use of MILNET. She indicated that net accounts for several E&V Team members had been recently established, and she polled the Team members to determine those who still needed accounts. She informed them that within approximately 1-2 weeks their accounts would be established. Then she provided a brief tutorial on how to access the net and how to use Hermes for message communication.

Jinny then explained that a new account has been established on ECLB, EV-INFORMATION, which will provide information to the public on the activities of the E&V Task. Anyone who has access to the net may log into EV-INFORMATION (password EV) to read the available files.

2.2 Common APSE Interface Set (CAIS)

Jack Kramer provided a presentation on the CAIS, and began his presentation by stating that the purpose of the development of the CAIS (initiated by a tri-service Memorandum of Agreement) is to facilitate tool portability and to provide standardization for tool interfaces. Jack indicated that acceptance of the CAIS will be dependent upon tri-service involvement as well as public involvement in its development. He then summarized the strategy behind the CAIS development which includes: (1) one standard set; (2) foundation based upon the ALS and AIE; (3) incremental development; (4) development of a validation capability (a priority item for E&V); (5) DoD maintenance; (6) evolutionary development; and (7) eventual transition of ALS and AIE.

Jack listed the CAIS characteristics: (1) simple, uniform underlying model; (2) noninterference with implementation strategies; (3) smooth integration with features of Ada; (4) flexible foundation for configuration management; and (5) merging of modern operating system and database management system ideas. He then provided the names (and organizations) of members of the

KIT/KITIA CAIS Working Group (CAISWG), pointing out that from the very onset, both Intermetrics (the AIE contractor) and SofTech (the ALS contractor) were participants. He listed the main areas of CAIS 1.0 (structures, files, processes, and devices) and stated that there are still deferred items such as configuration management, database management, multi-level security, and host-target relationships.

Jack proceeded to describe the concept of a node, indicating that the CAIS distinguishes between file nodes, device nodes, process nodes, and structural nodes. He included a graph which illustrated the node model concept. Then he went into detail on the concept of the process node. He stated that a process node represents the execution of an Ada program and all of its tasks and that a process node provides a common model for access to resources required to support the program execution. He also said that the process model supports the concurrent execution of Ada programs. Jack indicated that the time allocated for his presentation precluded a more detailed discussion of the CAIS. However, he offerred to provide such detailed discussions at future E&V meetings.

Jack concluded his presentation with a schedule of the CAIS development, emphasizing that the current CAIS would be improved as much as possible before being submitted for standardization in January, 1985. The CAIS would continue to be developed, with MIL-STD Version 2 available in January, 1987. Following Jack's presentation, one team member noted that feedback received from certain users and implementers indicated opposition to standardization of the CAIS because of run-time issues associated with target systems. The proposal was that a standard set of packages be developed for such run-time support. Jack also responded to an additional team member question on how tools actually reference CAIS packages.

2.3 E&V Working Group Review/Selection

Jinny Castor outlined the procedures by which documents are developed within the E&V Task. The initial phases are restricted to E&V Team members only: (1) draft document prepared within designated working group; (2) review of the draft document by the entire E&V Team; and (3) revision/refinement of the document by the designated working group. Once the document has been approved by the E&V Team it must then receive final approval from the E&V Team Chairperson and the AJPO Point of Contact. After

final approval has been received, the document (if available in file format) will be added to the EV-INFORMATION directory. The document or article (if appropriate) may also be submitted for publication once final approval has been obtained. All documents will be included within the annual E&V Public Report. A team member questioned whether it would be appropriate to include his organizational review of a draft document during the initial phase. The agreed upon answer was that such technical contributions were welcome, provided the draft document were treated with the sensitivity specified (i.e, no further distribution until final approval received).

During this discussion, Jinny clarified an issue which a team member had brought to her attention. There appeared to be a discrepancy between the AJPO initiated letter of Jun 83, which requested a tri-service management committee to supervise the efforts of the E&V Task. Jinny explained that at a subsequent Ada tri-service review, the service representatives opposed the use of a management committee, claiming that such representatives did not qualify as service managerial policy representatives. The E&V Task was designated as technical in nature, and required only technical direction from a Chairperson. Management issues would be addressed by the service representatives at the Ada tri-service reviews.

Jinny then reviewed the responsibilities of the five designated working groups and then polled all of the E&V team members as to which working group each member preferred to contribute. Following the selection of working groups, Jinny provided a list of responsibilities which would be assumed by each working group Chairperson: (1) representing the working group to the E&V Team; (2) serving as focal point for the working group activities; (3) conducting the working group sessions; and (4) coordinating all working group activities. She stated that the role of the working group Vice-Chairperson would be to assume the responsibilities of the Chairperson in his/her absence. Before the Team dispersed into the selected working groups, she requested that, following the working group meetings, each working group Chairperson report to the E&V Team the following information: (1) name of the Chairperson and Vice-Chairperson; (2) name of all working group members; (3) planned activities of the working group (action items to be accomplished by the next E&V Team meeting and anticipated presentations for the next E&V Team meeting); and (4) any other pertinent information. The E&V Team members then began to meet in separate working group sessions.

2.4 Working Group Reports

2.4.1 Technical Coordination Working Group (TECWG)

The TECWG report was given by Jimmy Williamson (Chairperson) who indicated that the Vice-Chairperson was Joe Genlot and that there were no other members currently on the TECWG. Jimmy stated that the activities planned for the TECWG were: (1) to perform a literature search to identify previous related technical efforts from which the E&V task could benefit; (2) to identify ongoing related technical efforts; and (3) to get "up-to-speed" on these related efforts. He indicated that the anticipated presentations for the Mar 84 meeting would include: (1) results of the literature search; and (2) identification of relevant areas.

2.4.2 APSE Analalyst Working Group (APSEWG)

The APSEWG report was given by Gina Burt (Chairperson) who indicated that the Vice-Chairperson was Angela Crumley and that the other members of the team included Georgeanne Chitwood, Dorothy John, Terry Humphrey, Liz Kean, and Rich Wallace. Gina stated that the activities planned for the APSEWG were: (1) to identify candidate APSEs for evaluation; (2) to create criteria for selecting certain APSEs for evaluation; and (3) to review documents for background information (such as STONEMAN, the AFWAL ROLM Evaluation Criteria Document, and the General Dynamics Update to the AFWAL ROLM Evaluation Criteria Document). She stated that the anticipated presentations for the Mar 84 meeting would include presentations on the following systems: ALS, AIE, ROLM, and Telesoft.

2.4.3 Public Coordination Working Group (PUBWG)

The PUBWG report was given by Chris Anderson (Chairperson) who stated that the Vice-Chairperson was Don Jennings and that the other member of the PUBWG was Pat Maher. She stated that the planned activities for the PUBWG by the Mar 84 meeting included the following: (1) initiation of the development of an E&V Project Reference List (using the documents provided at this meeting and including a brief synopsis of each); (2) establishment of procedures for team members to contribute to the E&V Project Reference List; (3) identification of professional organizations and journals; (4) establishment of procedures for

recording minutes and action items; (5) preparation of text for existing E&V viewgraphs for public presentations; (6) initiation of the development of a public coordination document (format to be established by Mar); and (7) establishment of procedures for team members to submit status reports for publication.

2.4.4 CAIS Working Group (CAISWG)

The CAISWG report was given by Nelson Estes (Chairperson) who stated that the Vice-Chairperson was Dave Fautheree and that the other member of the working group was Bob Harrell. In addition, Tim Lindquist was serving as technical consultant to the group. Nelson indicated that the first goal of the group was to try to understand the CAIS as it currently exists. In order to do so, the group would obtain a copy of the CAIS slides used during the CAIS public review in Sep 83 and determine what is and is not clear. That determination would then be funneled back to the KIT/KITIA. The CAISWG would also examine the current testing procedures used on the ALS and other systems in CAIS related areas. In particular, the CAISWG was interested in how the UNIX system is so easily transported from one system to another. The CAISWG is also responsible for the development of the Validation Procedures Document. The CAISWG members also planned to develop a check list of items to look for when the E&V Team eventually goes out for a contractual effort to develop the CAIS Validation Capability (CVC). Such items would be based upon the current CAIS 1.1 capabilities, expanded CAIS areas, and APSE component areas which represent an expansion to those currently provided in the NBS APSE Taxonomy Document.

2.4.5 Requirements Working Group (REQWG)

The REQWG report was given by Dan Burton (Chairperson) who stated that the Vice-Chairperson was John Miller and that the additional team members included Greg Bettice, Sam Dugan, Rich Fleming, and John Prentice. In addition, Jack Kramer was serving as technical consultant to this group. Dan stated that the planned activities for the REQWG included: (1) identification of the relationship of the E&V requirements to STONEMAN and establishment of contact with the KIT/KITIA Stoneman revision activity; (2) scoping the size of the E&V effort by getting an overall picture of the E&V Task; and (3) identifying an approach to development of the Requirements Document by increments. Dan also stated that the REQWG presentation at the Mar 84 meeting

would provide the overall approach to development of the Requirements Document, with emphasis on the first increment.

2.5 Open Discussion

Jinny Castor explained to the E&V Team members that AFWAL has been assisting the Army (CENTACS at Fort Monmouth) in testing of the ALS and has thus far been the only Air Force organization with access to the ALS. However, based upon agreement with the Army, AFWAL is allowed to distribute the ALS to other Air Force organizations, provided such organizations are assisting AFWAL in its testing procedures. With this background information, Jinny then stated that Air Force organizations represented on the E&V Team could obtain copies of the ALS from AFWAL. She stated that in order to obtain a copy of the ALS the organization must send an official request to her stating the following: (1) that the organization is represented on the E&V Team; (2) that the organization is requesting the ALS for the purpose of providing testing support only (i.e., no production development); and (3) that the organization will not release its copy of the ALS to a third party. In response to that request, Jinny will send a letter which specifies the requirements for the tapes to be sent to AFWAL.

The discussion of the ALS requirements was then expanded upon by Rich Wallace who stated that the ALS runs on VAX/VMS Version 3.1. It requires 4 megabytes of memory and at least 200 megabytes of additional storage. The ALS installed without any database requires 10 megabytes, and with the testing database requres 80 megabytes. The ALS requires space on both the VMS system disk and a user disk (which may create problems for potential users).

Jinny then provided information on the scheduled E&V Workshop to be held in Airlie, Virginia 2-6 Apr 84. She indicated that reservations had been made for 25 individuals, approximately 10 of which would be E&V Team members/consultants. She said that a CBD article would be published to solicit position papers and candidates from industry and academia. The working group chairs were invited to participate (registration fee approximately $140 to cover meals and facilities, and room rates of approximately $32 per night). She requested that all working group chairs respond by the Mar 84 meeting as to whether or not they would attend, and if not, the vice-chair would have the opportunity to attend.

During the final discussion, the question was asked as to whether or not E&V meetings would be held in conjunction with AdaJUG/AdaTEC meetings. Jinny stated that all E&V meetings would be held at Wright-Patterson Air Force Base and that all Air Force organizations had made the commitment to 4 TDYs per year for participation in the E&V Task. Brian Schaar then expanded upon this discussion, stating that E&V was under the auspices of the AJPO, and that although E&V coordination with professional organizations was encouraged, no direction to the E&V Task was given by any other than AJPO. Brian also emphasized the need for close coordination between E&V and KIT/KITIA. Brian and Jinny agreed that a meeting with Tricia Oberndorf (KIT Chairperson) would be beneficial to establish the means by which such coordination could be accomplished.

The first E&V meeting was then adjourned.

List of Attendees

Anderson, Chris (AFATL/DLMM)
Bettice, Greg (NAC)
Burt, Gina (AFALC/PTEC)
Burton, Dan (ESD/ALL)
Castor, Jinny (AFWAL/AAAF)
Chitwood, Georgeanne (ASD/ADOL)
Crumley, Angela (AFWAL/AAAF-2)
Dobbs, Paul (General Dynamics)
Dugan, Sam (SA-ALC/MMEC)
Estes, Nelson (ASD-AFALC/AXTS)
Fautheree, Dave (AFCMD/KRS)
Fleming, Rich (Aerospace Corp)
Genlot, Joe (CENTACS)
Harrell, Bob (AFCCPC/SKXX)
Humphrey, Terry (NASA)
Jennings, Don (OC-ALC/MMECE)
John, Dorothy (AFWAL/AAAF-1)
Kean, Liz (RADC/COES)
Knapper, Bob (IDA)
Kramer, Jack (IDA)
Lindquist, Tim (VPI)
Maher, Pat (OO-ALC/MMECF)
Miller, John (SM-ALC/MMEHP)
Prentice, John (AFHRL/IDC)
Schaar, Brian (AJPO)
Wallace, Rich (AFWAL/AAAF-2)
Williamson, Jimmy (AFWAL/AAAF-2)

APPENDIX G

MINUTES

of the

EVALUATION & VALIDATION (E&V) MEETING

7 - 8 MARCH 1984

Wright-Patterson Air Force Base, Ohio

Table of Contents

Minutes
E&V Meeting
7-8 Mar 84

1. Wednesday, 7 March 1984

1.1 Welcome and Introductions

The E&V meeting began with a welcome by the Chairperson, Jinny Castor, followed by an announcement of agenda changes. The list of E&V Team members with addresses and telephone numbers was circulated in the audience for updating purposes. The minutes of the first meeting were approved and are available to the public in the EV-Information directory.

The format for the minutes has been reviewed and finalized. Attendees seated at the head table were introduced: LCDR Brian Schaar (Ada Joint Program Office), Kevin Chadwick (Canadian National Defense Headquarters), Maj Izzy Caro (Air Force Wright Aeronautical Laboratories), Jack Kramer (Institute for Defense Analyses), Tricia Oberndorf (Chairperson of KAPSE Interface Team), Tim Lindquist (Virginia Polytechnic Institute), Betsy Bailey (Institute for Defense Analyses) and Ronnie Martin (Georgia Institute of Technology). Each Team member then provided a brief self-introduction.

Representatives between the KIT and the E&V Team were identified: Jinny Castor, Kevin Chadwick, Liz Kean, Jack Kramer, Tim Lindquist, Lucas Maglieri, Tricia Oberndorf, Brian Schaar and Guy Taylor.

1.1.1 E&V KIT/KITIA Coordination Strategy

Jinny Castor and Tricia Oberndorf have developed a coordination strategy between the E&V and KIT/KITIA teams. Jinny Castor will be presenting Quarterly E&V Status Reports at the KIT and KITIA meetings. Tricia Oberndorf or her representative will be presenting the status of the KIT and KITIA at the E&V meetings. The common representatives between teams will serve as channels of communication for issues or questions that may arise.

1.2 KIT/KITIA Status

Tricia Oberndorf presented background information and the current status of the KIT/KITIA. The KIT was established by a Tri-Service Memorandum of Agreement in January 1982. The Navy was designated as the lead organization. The objective of this Team is to establish common interfaces, so that tools and data bases can be shared between Ada Programming Support Environments (APSEs). An auxiliary team with members from industry and academia was established, the KIT from Industry/Academia (KITIA), to provide additional expertise. Currently there are 30 members of the KIT and 30 members of the KITIA.

Sharing tools and data bases implies tools must be interoperable and transportable. Transportability is defined as the ability of a tool to be moved from one APSE to another without reprogramming and to maintain the identical functionality. Since 100% transportability is not possible, transportability is measured by the degree in which it can be achieved without tool reprogramming. Interoperability is defined as the ability of APSEs to share data bases, including objects and their relationships, without modification. Again, interoperability cannot be 100% achieved. It is measured by the degree to which it can be accomplished without data base modification.

KIT/KITIA developing products include the Common APSE Interface Set (CAIS). The CAIS is a set of interfaces. CAIS version 1.1 is predominately on the KAPSE level focusing on the transportability issue. Interoperability will be addressed in later CAIS versions. It is anticipated that the CAIS will eventually become a Mil-Standard.

Other important KIT/KITIA developing products include the requirements and design criteria (RAC) document. The document will contain the requirements to which the CAIS version 2.0 (the final CAIS) must comply. The Guideline and Conventions (GAC) document will provide "I&T Suggestions" to future KAPSE implementors and tool writers. The KIT/KITIA Public Report, published approximately every six months, is available through NTIS. These reports provide a comprehensive update for the reporting period.

The KIT/KITIA schedule is as follows: RAC - approved baseline (April 84); CAIS - version 1.1 (Sep 83), version 1.2 (April 84), version 1.3 (Nov 84), Mil-STD 1 (Jan 85), initiate version 2.0 (Jan 85), version 2.0 draft (Jan 86), Mil-STD 2 (Jan 87); GAC - initiate draft (Jan 85), final (Jan 87); Public Reports - approximately every six months.

Another member of both the E&V Team (alternate member) and KIT was noted: Larry Lindley from the Naval Avionics Center.

A proposal has been made to the Tri-Service Review that CAIS version 2.0 be pursued in a dual mode (i.e., parallel competitive procurements) culminating in a "fly-off." To date, this proposal has not been approved or disapproved.

There are seven named KIT/KITIA working groups: CAISWG - CAIS development; STONEWG - STONEMAN refinement and strengthening; DEFWG - KIT/KITIA glossary development; STANDWG - existing standards examination, and standardization process guidance; COMPWG - CAIS semantic definition facilitating validation; RACWG - development of the RAC and an accompanying rationale; GACWG - development of the GAC.

1.3 Toward a Specification Technique for CAIS

Dr. Tim Lindquist, VPI, presented an overview of an effort sponsored by the AJPO through the office of Naval Research Information Sciences Division. The researchers include Dr. Lindquist, Dr. D. G. Kafura and Mr. J. Facemire. The effort involves using an abstract machine approach in specifying the CAIS.

In order to validate a CAIS and in order for tool writers to interface with the CAIS, the CAIS specification must be precise (i.e., complete, consistent, concise, and unambiguous), usable by specifiers, implementors and users, and must lead to some type of validation mechanism.

Currently the CAIS specification consists of package specifications for the types and operations that make up the CAIS and associated commentary. A commentary excerpt from the Process Section of the CAIS was presented to illustrate a few problems with this method of CAIS specification.

Due to incomplete and inconsistent lexicographic phrasing in the sample commentary, a CAIS validator would have great difficulty in trying to construct test cases. The example pointed out the need for a more precise specification to understand the semantics of the CAIS. From the current CAIS specification, it would be extremely difficult to develop a validation suite that could distinguish between different implementations of CAIS without the validator interpreting semantics.

Dr. Lindquist then presented four aspects considered essential in specifying software products (and the CAIS in particular): (1) Syntax - adequately described in the CAIS using package specifications; (2) Functionality - functionality of operations. (3) Protocols and Hidden Interfaces - the interface between the operations. These are the keys that tell the validator how to construct a validation suite and tell the user how to use the CAIS. (4) Limits - pragmatic limits on implementations.

Dr. Lindquist then explained in more detail the aspects considered essential to the CAIS specification and outlined a specification mechanism that incorporates all of these aspects, thus facilitating the validation process.

Since the syntax aspect is currently well described in the CAIS, Dr. Lindquist began with the functionality aspect. Several approaches to describing CAIS functionality were presented. The formal approach provides a complete method for describing semantics as long as there exists an operational definition. Since the CAIS has no operational definition, the formal approach to describing functionality must be ruled out.

The commentary approach to describing functionality is currently used in the CAIS specification. This approach has the advantage of ease of writing but has serious shortcomings due to resulting inconsistent and incomplete descriptions which make validation difficult.

Another approach to describing CAIS functionality is by example. The public review overheads provide an example of this approach.

The last approach to describing CAIS functionality is the abstract machine method. This is the subject of Dr. Lindquist's research and the remainder of his talk was devoted to it. An abstract machine includes a storage environment, a set of instructions and a processor. A processor interprets an instruction in the context of a storage environment. To fully define an abstract machine that can be used to define the functionality of the CAIS, the storage, instructions and processor must be fully defined. In his abstract machine model, for storage any kind of Ada object is allowed. An extension of Ada's data facilities is defined to declare axiomatic objects so that the abstract machine can store objects of axiomatic types. Instructions that the abstract machine recognizes are Ada language instructions. Thus, the processor for the abstract machine is able to interpret Ada instructions and extended Ada instructions. Ada is a good choice for an abstract machine description of the CAIS due to its richness, thorough definition and compatibility with the problem environment. The specification does not have to be executable but it must be readable -- one of Ada's strong points.

Several abstract machine examples based on the current version of the CAIS were presented.

A question from the audience brought out the point that the KIT/KITIA charter focuses on Stoneman - like APSEs and not target environments. The AJPO is reviewing the need for runtime environment standardization but KIT/KITIA currently is not addressing this issue.

Besides functionality, other aspects to consider in the specification of the CAIS include protocols and hidden interfaces. An interaction between a CAIS operation and a tool is defined as a hidden interface. An interaction between two different CAIS operations is defined as a protocol. Two types of CAIS protocols were identified: Uses Hierarchy - CAIS functions needed by CAIS operations. The abstract machine description makes these explicit; Use of Underlying Model - different CAIS facilities can interact with each other by the data that they produce in this underlying model of an implementation.

In order to explain hidden interfaces properly, Dr. Lindquist went back to the CAIS. The CAIS, itself, is an interface. It has two interfaces: above and below. The above interface is with the tools. The implementation of the CAIS also has an interface below to the underlying operating system, hardware, etc. Below interfaces must be specified as well as the above ones in order to be able to validate the CAIS. These below interfaces are called hidden because they aren't direct.

The last aspect that is essential to properly specifying the CAIS is pragmatic limits. There are two types of limits: use limits and value limits. Use limits refer to a tool's use of a CAIS facility. Value limits refer to the parameters that are passed to CAIS operations. In order to validate, the limits must be specified in an amenable way to validation (e.g., use Ada package STANDARD concept). As of now at VPI, there exists a preliminary set of abstract programs that describes all of the CAIS node model. Refinement of these programs is continuing.

1.4 Software Test and Evaluation Project (STEP)

Ronnie Martin, Georgia Institute of Technology, presented an overview of the STEP effort.

1.4.1 Background

STEP was initiated by the Director Defense Test and Evaluation (DDT&E) in 1981. The primary goal of STEP is to improve the practice of test and evaluation for mission critical software. Current STEP efforts are concentrated in three areas: development of policy and standards; insertion of existing technology; and coordination with related efforts.

The primary function of DDT&E is to assess the maturity of major systems under development and provide recommendations to the DSARC as to whether or not the systems should be allowed to pass the DSARC milestones. The criticality of software to the successful accomplishment of DoD missions and therefore to the acquisition decision making process is increasing. The imbalances in the way hardware and software are developed and tested have been documented. In 1974, the Defense Science Board produced a report stating that in hardware development, the hardware was carefully developed and tested with many milestones. This was not the case in software developments. Further aggravating the lack of milestones is the fact that testing is expensive and therefore avoided. However, the escalation in the cost of finding and fixing software errors as time passes intensifies the need for effective "up-front" testing. Finally, although DoDD 5000.3 (Test & Evaluation) has a specific section addressing software test and evaluation, it is vague and inadequate. It was for these reasons that STEP was established.

1.4.2 STEP Approach

The first two years of the STEP project were spent gathering and analyzing information to determine the feasibility of improving policy at the DoD level. This feasibility study resulted in twenty recommendations. One thrust of the recommendations was in the area of test planning. It was recommended that tests be planned to form a chain beginning at the system level and continuing through the unit or module level such that each level of testing relates to and supports the previous level. Another thrust of the recommendations was to focus software testing on critical software components in order to use scarce resources in the most effective manner possible. Other recommendations addressed the need to insert available software testing technology into practice.

1.4.3 State-of-the-Art Findings

There are two categories of software testing methodology: static analysis and dynamic analysis. Static analysis of software is that analysis that does not require the execution of code (e.g., design walkthroughs, structured design reviews). Dynamic analysis of software is that analysis that does require the execution of code.

Ronnie Martin outlined several testing techniques along with the advantages, disadvantages and the size of the software (small, medium, or large) to which they could be applied. In general, the static methods were found to be effective in terms of locating errors in the design and code. Most dynamic methods require the use of automated tools.

1.4.4 State-of-the-Art Practice Findings

Interviews were conducted at organizations in all three services and Defense contractor facilities involved in mission critical software testing. One finding was that the military acquisition organizations seemed to have complete faith in their contractors. Reasons for this include a lack of manpower, and a lack of software expertise. Another finding was that after the critical design review (CDR) occurs, the next software milestone is acceptance testing. More milestones are necessary for the military to track the software development process. Another finding indicated that contractors put complete faith in the programmers. Audit procedures to check conformance to organizational standards for software development and testing rarely exist.

In order to determine the state-of-practice regarding testing tools, information was requested from identified contacts for 218 tools which were found in open literature. Only 42 responses were received. The findings were as follows: Most tools are tailored to specific projects. The amount of money required to package a tool for commercial use can be nine times that of the development cost, thus, discouraging this activity. Few tools are suitable for mission critical applications. No cost benefit studies exist to justify the use of testing tools to management. Few comparative studies exist for testing tools.

1.4.5 Recommendations

Based on the study results, the following recommendations were made: modify DoDD 5000.3 to require system level planning for mission critical software, specify the required level of tests, and require test reporting; conduct tests throughout the lifecycle; develop requirements document and test plan document at the beginning of the program; interpret test results in terms of system objectives; define and meet specified evaluation criteria; and balance risk of hardware with software. Other recommendations were concerned with initiating a technology upgrade and improving the application of existing technology.

1.4.6 STEP and the E&V Effort

Certain issues in the Test & Evaluation Master Plan (TEMP) directly relate to what is being addressed in the E&V effort. Evaluation & Validation of testing tools will help DDT&E better determine the risk of using these tools for system development. Also, STEP is developing the requirements for an APSE test environment. This environment will need to be evaluated and validated.

1.4.7 Improvements

DDT&E is already following a draft STEP checklist in evaluating software in system acquisitions. The STEP Phases I and II Final Reports are available through DTIC or through the Georgia Institute of Technology at a cost of $50.00.

1.5 E&V Working Group Presentations

1.5.1 REQWG Report

The Requirements Working Group (REQWG) report was presented by Major Dan Burton, the Chairperson. The REQWG approach is in three steps: develop an outline for the requirements document; fully develop several sections of the document that are most familiar to the working group members; develop the remaining sections of the documents. Several candidate sections for the E&V Requirements Document were listed: E&V Product Quality, Specific APSE Tool Requirements, Usability of E&V, and APSE Considerations.

1.5.2 TECWG Report

The Technical Working Group (TECWG) report was presented by Jimmy Williamson, the Chairperson. The TECWG performed a literature search for related E&V documents. No previous related efforts were identified. Several potentially E&V related ongoing efforts have been identified: Software Test and Evaluation project (STEP), STARS Measurement project, STARS Research in Software Reusability, Communications Software Technology Program (CSTP), Common APSE Interface Set (CAIS), Ada Language System (ALS), and Ada Integrated Environment (AIE). A proposed TECWG Strategy Document format was presented for review by the team.

1.5.3 PUBWG Report

The Public Coordination Working Group (PUBWG) report was presented by Chris Anderson, the Chairperson. A strawman document of the Public Coordination Strategy Document (final due in June), was distributed to the team for review and comment.

The draft Project Reference List was also distributed. Several mechanisms for transitioning E&V technology to the public were presented: briefings, publications, E&V Quarterly Report, EV-Information net directory, Project Reference List, and the E&V Public Report. Several publications and organizations were identified as candidates for E&V technology presentations. A recommendation was made to brief E&V status at every AdaJUG and AdaTEC meeting. The E&V Chairperson tasked the team to review the PUBWG strawman document by 30 March.

1.5.4 CAISWG Report

The CAIS Working Group (CAISWG) report was presented by Nelson Estes, the Chairperson. A brief overview of the CAIS was given. Nelson pointed out that the current CAIS guarantees transportability, not interoperability of tools, but that this will change as the CAIS matures.

Tricia Oberndorf pointed out that the name CAIS, Common APSE Interface Set was chosen very carefully. Any and all standardized levels in the APSE that affect interoperability and transportability (I&T) are potential parts of the CAIS (e.g., if DIANA is standardized, it could become part of the CAIS).

1.5.5 APSEWG Report

The APSE Working Group (APSEWG) Chairperson, Lt Rick Long, introduced five APSEWG speakers.

1.5.5.1 ALS

The Ada Language System (ALS), developed by SofTech for the Army, was discussed by Rich Wallace. Rich presented his perception of the ALS implementation. He pointed out that it diverges somewhat from the Stoneman APSE. According to Rich, the ALS has been implemented on a shaft and base of VMS, thus it is not a true independent environment. Rather, it is an application program on VMS. A list of ALS tools was discussed. ALS tools can be grouped into the following categories: Advanced Configuration Control (not available yet), Configuration Control, Command Language Processor, Data Base Manager, Display and File Administrator. The current version of the ALS uses the DEC VAX-11 EDT text editor. SofTech can distribute this under a license agreement with DEC. Currently, tools are written in Ada '81. Conversion to Ada '83 is scheduled for completion in July 1984. Public release date for the ALS is December 1984.

1.5.5.2 ALS/N

The Navy's version of the ALS was presented by Guy Taylor. The Navy is planning to build Navy unique tools to add to the ALS. Currently, proposed targets include the AN/UYK-44 (16-bits), AN/AYK-14 (16-bits), and AN/UYK-43 (32-bits). Due to funding problems, the ALS/N has been indefinitely postponed with the exception of the AN/UYK-44. SofTech is performing the retargeting effort. The program performance specification and program design specification have been delivered. The code generator, runtime library, loader, importer and exporter will be delivered in April 1985.

1.5.5.3 AIE

The Ada Integrated Environment (AIE), being developed by Intermetrics for the Air Force, was presented by Liz Kean. Due to funding problems, work was stopped on all tools except the compiler. The compiler's host and target is the IBM 4341 using the UTS operating system. The compiler uses a modified version of the Tartan Laboratory's DIANA. No date has been set for resuming other tool development. When it is resumed, the following components are planned as part of the environment: KAPSE, database manager, Editor, Debugger, Program Library Manager, Linker, Command Language Processor, and Compiler. When the APSE effort resumes, the compiler will be rehosted on the KAPSE. The Ada compiler on UTS is scheduled for delivery in January 1985.

1.5.5.4 ROLM Ada Work Environment

The ROLM Ada Work Environment was presented by Georgeanne Chitwood. The Ada Work Environment is a complete configuration that includes hardware, software and support. The hardware centers on the 32-bit ECLIPSE MV/Family. The Ada compiler accepts the ANSI/Mil-Std-1815A - 1983 version of the language. The Environment also includes an Ada Programming environment which includes: a "KAPSE" (ROLM has plans to interface to CAIS when it is fully defined), a user interface (command language interpretor), data base control tools, application development tools, target development tools and libraries of packages. The Language Control Facility (LCF) at Wright-Patterson has a ROLM Environment with 8 terminals. Georgeanne pointed out that Ada training is not being performed by the LCF at this time.

1.5.5.5 Telesoft Ada Programming Support Environments

The Telesoft Environments were presented by Terry Humphrey. The current systems operate on the VAX 11/780, MC68000, IBM 370, INT 370, IBM PC or XT, and INT 86. Support tools include: KAPSE, Ada compiler (currently not full Ada, generates P-code), linker/executor, runtime library, text I/O, screen editor, embedded system kit (helps user to customize runtime level on MC68000), and computer-based learning materials. Telesoft's new compiler (certified, with the first validation scheduled for mid-March) will generate native code for the VAX 11/780, IBM 370, MC68000, and INTEL 8086/88. Host and targets in all cases are the same.

1.6 Open Discussion

During the open discussion, various issues were discussed including TAC registration, ECLB accounts for new members, request for team input to E&V presentations, and AFWAL ROLM System availability to Wright-Patterson E&V team members for E&V related activities.

Industry responses to the E&V Workshop announcement in the CBD were received. Invitations to 21 industry representatives have been sent. The Workshop will be organized into three working groups: Requirements - Tim Lindquist, Chairperson; Reference Manual - Jack Kramer, Chairperson; and Recommendations - Chris Anderson, Chairperson.

Jinny Castor will give E&V presentations at the National Security Industries Association (NISA) on 23 May and at the next AdaTEC and AdaJUG meetings.

Betsy Bailey will be conducting a Human Factors Workshop in May. It is jointly sponsored by AJPO and STARS. One of the objectives is to investigate human factors as they apply to the evaluation of current APSEs. Betsy will present the results of the workshop at the June E&V meeting.

The RFP for the technical support contractor to the E&V team will be going to Procurement shortly.

The E&V effort will be developing a terminology list. The KIT/KITIA terminology list will be used as a basis for the E&V list.

The E&V meeting was adjourned for the day.

2. Thursday, 8 March 1984

2.1 Announcements/Discussion

Jinny Castor opened the second day of the E&V meeting. A discussion of several issues was conducted. It was decided to expand future E&V meetings to two and one-half days. Regarding team net communication, it was decided that messages should be sent to the appropriate group with a copy to the team with a very descriptive title.

2.2 Working Group Status Reports

2.2.1 PUBWG Status Report

The PUBWG Status report was given by Chris Anderson (Chairperson). No personnel changes occurred in the PUBWG. Draft copies of the Project Reference List, Public Coordination Strategy document, organizations list and publications list were distributed to the team for comment. Projected work for next quarter includes putting the Project Reference List and PUBWG forms in the EV-Information directory. The minutes and condensed minutes (E&V Quarterly Report) will be completed. The first version of the Public Coordination document will be delivered at the June meeting.

2.2.2 REQWG Status Report

The REQWG Status report was given by Tim Lindquist. New members in the REQWG include Betsy Bailey, and Ronnie Martin. No deliverables were due this quarter. REQWG accomplishments include the strawman outline for the Requirements document. Deliverables for next quarter includes the first version of the Requirements Document. Betsy Bailey will be giving a presentation at the June meeting on the Human Factors Workshop. A revised (as a result of working group meetings) strawman outline of the Requirements Document was presented. The outline includes five sections: General Introduction, Approach, Product Quality Guidance, APSE E&V Requirements, and Summary/Conclusions.

2.2.3 TECWG Status Report

The TECWG Status report was given by Jimmy Williamson (Chairperson). Personnel additions to the group includes John Taylor, Mark Mears and Kevin Chadwick. Joe Genlot, the Army representative, was unable to continue as a Team member. TECWG accomplishments this quarter include a literature search, and TECWG Coordination Strategy Document format. Projected work for next quarter includes a follow-up to the E&V related efforts identified, a draft of the Technical Coordination Strategy document, and development of a matrix, listing E&V team members with related E&V activities in which they are engaged. The objective of the matrix is to identify focal points for coordination of E&V activities with other related efforts.

2.2.4 CAISWG Status Report

The CAISWG Status Report was given by Nelson Estes (Chairperson). Darlene Avery is a new member of the CAISWG. Lori Macky (Bob Harrell's replacement) may also join the group. The CAISWG will begin development of a group of tests for each version of the CAIS as it becomes available: CAIS 1.1, CAIS as it is reflected by the E&V REQWG, CAIS-Mil-Std 1.0, and the CAIS as it becomes apparent from the KIT Implementors Guide. The CAISWG also plans to provide samples of Ada code that use the CAIS (i.e., CAIS benchmarks).

2.2.5 APSEWG Status Report

The APSEWG Status report was given by Rick Long (new Chairperson). Guy Taylor is also a new member. Deliverables due this quarter were the presentations on the following systems: AIE, ALS, ALS/N, ROLM and Telesoft. Projected work for next quarter includes a list of tools in various APSEs. This is an intermediate step in defining basic APSE functions. The APSEWG also plans to maintain a "lessons learned" file based on the use of various APSEs.

2.3 Open Discussion

Jinny encouraged working group members to sit in on other working group sessions when possible. Publication of position papers by team members was also encouraged. A document was called to the attention of the team: "Guideline: A Framework for the Evaluation and Comparison of Software Development Tools," published by the National Bureau of Standards. The REQWG will be sent copies for review.

The E&V meeting was then adjourned.

ATTENDANCE LIST
E&V Team Meeting, 7 - 8 Mar 84

LCDR B. Schaar
Ada Joint Program Office
Wash DC

AFWAL/AAAF (J. Castor)
Wright-Patterson AFB OH

AFWAL/AAAF (Maj I. Caro)
Wright-Patterson AFB, OH

AFWAL/AAAF-2 (J. Williamson)
Wright-Patterson AFB, OH

AFWAL/AAAF-2 (Lt R. Wallace)
Wright-Patterson AFB, OH

AFWAL/AAAF-2 (Lt R. Long)
Wright-Patterson AFB, OH

AFWAL/FIGRB (Lt D. Avery)
Wright-Patterson AFB, OH

AFWAL/FIGLB (M. Mears)
Wright-Patterson AFB, OH

AFALC/PTEC (G. Burt)
Wright-Patterson AFB, OH

AFATL/DLMM (C. Anderson)
Eglin AFB, FL

AFCMD/SI (Lt D. Fautheree)
Kirtland AFB, NM

AFHRL/IDC (Lt J. Prentice)
Lowry AFB, CO

AFLC/MMEC (Capt J. Taylor)
Wright-Patterson AFB, OH

ASD/ADOL (G. Chitwood)
Wright-Patterson AFB, OH

ASD-AFALC/AXTS (N. Estes)
Wright-Patterson AFB, OH

OO-ALC/MMECF (P. Maher)
Hill AFB, UT

RADC/COES (E. Kean)
Griffiss AFB, NY

SA-ALC/MMEC (H. Dorsett)
Kelly AFB, TX

SM-ALC/MMEHP (W. Happ)
McClellan AFB, CA

R. Fleming
Aerospace Corporation
Los Angeles, CA

NOSC (T. Oberndorf)
San Diego, CA

NAC (G. Bettice)
Indianapolis, IN

NAC (L. Lindley)
Indianapolis, IN

FCDSSA (G. Taylor)
Virginia Beach, VA

R. Martin
Georgia Institute of Technology
Atlanta, GA

Johnson Space Ctr (T. Humphrey)
Houston, TX

K. Chadwick
National Defence Headquarters
Ottawa, Ontario

J. Kramer
Institute for Defense Analyses
Alexandria, VA

Betsy Kruesi Bailey
Institute for Defense Analyses
Alexandria, VA

AFCCPC/SKXX (R. Harrell)
Tinker AFB, OK

ESD/ALL (Maj D. Burton)
Hanscom AFB, MA

OC-ALC/MMECE (D. Jennings)
Tinker AFB, OK

T. Lindquist
Virginia Polytechnic Institute
and State University
Blacksburg, VA

APPENDIX H

MINUTES

of the

EVALUATION & VALIDATION (E&V) MEETING

6-8 JUNE 1984

Wright-Patterson Air Force Base, Ohio

Table of Contents

1.0 Wednesday, 6 March 1984

1.1 Welcome and General Business

The E&V meeting began with a welcome by the Chairperson, Jinny Castor, followed by the distribution of the official set of E&V viewgraphs. Team members were encouraged to brief their home organizations, so that they can better understand the E&V Task. A list of files on the <EV-INFORMATION> directory at USC-ECLB was also distributed.

Tom Griffin, the new E&V Support Administrator, was introduced, followed by self-introductions of all E&V Team Members.

Highlights of the E&V Workshop held in Arlie, Virginia from 2-6 April, 1984 were presented by Jinny Castor. Draft E&V Workshop documents were distributed to Team members for review. The goals of the Workshop were to develop draft documents concerning the Requirements, the Reference Manual and Recommendations. The final Workshop Report will be published and available to the public by September 1984. Industry Workshop participants will be invited to the September E&V meeting.

E&V presentations will be given at the next AdaJUG (16-18 July) and AdaTEC (30 July - 1 August) meetings. There will also be an E&V Birds-of-a-Feather (BOF) session at the next AdaTEC meeting.

1.2 Evaluating APSE Usability

Dr. Elizabeth Kruesi Bailey from the Institute for Defense Analyses gave a presentation concerning evaluation of Ada Programming Support Environment (APSE) usability. She emphasized the fact that APSEs are intended to support the software activities of people.

Dr. Bailey began by discussing three major points. (1) APSE Usability has a number of individual components that can be evaluated quantitatively, objectively and vigorously. (2) Precise definitions of usability is essential to the evaluation process. (3) Usability can be measured as quantitatively as any other system characteristic provided the focus is on the user's performance and subjective reactions and not on APSE features.

Dr. Bailey then outlined the following items to be discussed during the remainder of her presentation: the requirements for the evaluation of a system; an example of an evaluation of text editors; macro-level issues (i.e., at what level of the user's activities should the evaluation focus be placed); user characteristics; and finally combining all these factors together to define a general approach to evaluating the usability of APSEs.

1.2.1 Evaluation Requirements

The first requirement for the evaluation of an APSE is objectivity. Objectivity implies that the evaluation should be impartial (i.e., not biased toward any particular set of design constructs or design features).

The next requirement is thoroughness which implies the consideration of multiple aspects of the system being evaluated.

Other requirements are that characteristics evaluated must be relevant, and that the evaluation results must be replicable.

Finally, the evaluation process must be standardized so that results from independent evaluations can be compared.

1.2.1.1 Objectivity

Dr. Bailey discussed two approaches to achieving objectivity in the evaluation of APSEs. The first approach is to analyze the individual APSE features. However, given the current status of knowledge concerning APSE features, this approach would be highly subjective. Additionally, an evaluation of individual APSE features would not allow for the synergistic advantages that may be gained by combining certain APSE features.

A second approach is to focus the evaluation on user activities that APSEs are designed to support.

1.2.1.2 Thoroughness

Thoroughness of an APSE evaluation implies the consideration of multiple aspects of the system. These aspects include: ease of learning, retention and use for the beginning or intermittent user; efficiency and power as seen by the expert user; and user satisfaction for the discretionary user.

1.2.1.3 Relevance

Relevance of an evaluation implies that important APSE characteristics are being evaluated. Determining relevance requires competency in human performance measurement (i.e., experimental psychologist), attitude assessment (i.e., clinical psychologist) and the software development process (i.e., computer scientist).

1.2.1.4 Replicability

Replicability of an APSE evaluation implies that the same pattern of results can be obtained by independent evaluations. In order to achieve replicability the evaluation process must be precisely described in such a way that it is repeatable. This can best be accomplished via operational definitions where the characteristic being evaluated is defined by the operations that are used to conduct the evaluation.

1.2.1.5 Standardization

The fifth and final requirement is that the evaluation be standardized. This represents a more stringent form of the requirement for replicability by ensuring that independent evaluation results for different tools can be directly compared regardless of time elapsed between evaluations or evaluator. The requirement that an evaluation be standardized has a critically important side effect: it allows for the accumulation of evaluative information in the form of a central database.

1.2.2 An Example: A Comparative Evaluation of Text Editors

Dr. Bailey discussed an evaluation of text editors carried out by Roberts and Moran which meets the requirements previously discribed. The Roberts and Moran study involved a comparison of nine text editors. A set of fifty three benchmark editing tasks were defined in order to provide the common ground for comparison across editors. The evaluation encompassed the following three aspects of usability: ease of learning (time to complete tasks by novice users); efficiency (time to complete all tasks by experts); and errors (error correction time divided by error-free time by experts).

This study is of interest because it conforms to the requirements listed for an evaluation methodology. It is objective in that it focuses not on particular design features of editors but on the performance of the user in carrying out typical editing tasks. It is thorough in that multiple aspects of usability were examined. It is relevant in that the measure used conforms to general intuition about the characteristics that contribute to usability. It is replicable in that the procedures for conducting the evaluation are laid out in sufficient detail to enable an independent evaluation. Finally, it provides the basis for a standardized evaluation since all of the benchmark tasks are described in detail.

Dr. Bailey pointed out that only the most important results reported by Roberts and Moran was that the entire effort was doable and yielded meaningful results that are of interest to potential users of these editors.

1.2.3 Macro-level Issues in Evaluating Usability

Dr. Bailey next discussed the issue of defining the appropriate level at which APSE support for user activities should be evaluated. In examining different types of APSEs, there are changes in the integration of APSE functions, specific tools and lower-level user activities (e.g., compiling as a separable user activity disappears). What remains constant is the high-level activity (e.g., coding). Thus, this appears to be the proper level to focus the evaluation in order to achieve independence from APSE design constructs. The set of functions within an APSE are likely to change as more and more of the life-cycle becomes automated. Although more user functions will be supported, the user functions themselves, will not change.

1.2.4 User Characteristics

Next, Dr. Bailey discussed the need to characterize the APSE users. This is important because user characteristics can have major consequences for the components of usability. For example, ease of learning may be critically important to the first time user. Suggested dimensions of user characteristics include: job type, degree of discretion, frequency of use, expertise in system usage, expertise in job and team relationship.

1.2.5 A General Approach to Evaluating Usability

Dr. Bailey outlined the following steps in evaluating APSE usability. (1) Identify high-level user activities which will provide a common ground for evaluation. (2) Characterize the users. (3) Construct a matrix showing activities and users in order to point to components which should be emphasized. (4) Develop operational definitions for various components of usability within the context of each activity.

1.2.6 Policy Issues

Dr. Bailey pointed out the importance of publishing the evaluation procedures and results. Publication of the procedures will encourage APSE builders to use these tests during the design process. Publication of the results will eliminate repetitious evaluations on the same tools.

1.3 Johnson Space Center (JSC) APSE Project

Dr. Charles McKay from the University of Houston - Clear Lake (UH-CL) discussed the Johnson Space Center's effort to create a distributed network testbed so that empirical research related to space stations can be conducted. Dr. McKay said that using Ada as the programming design language would guarantee a higher probability of success due to Ada's capability to support parallelism, concurrency and fault tolerance.

As part of the education process to begin work on the testbed, a joint NASA/JSC UH-CL Steering Committee on software engineering with Ada, was formed. Membership includes personnel from JSC, UH-CL, and JSC contractors. The group meets regularly to discuss the work going on with respect to the following issues: models - distributed network, open systems interconnection, distributed relational database; methodologies - object oriented design, layered design; and tools - MAPSEs and beyond.

In developing the joint NASA/JSC UH-CL testbed, many factors must be considered: the workbench, the work - verification (KIT/KITIA related work), validation (develop a Network Operating System in order to demonstrate Ada's capabilities), and NASA Ada transition planning; and the Schedule - a 17 year log book.

Furthermore, the requirements for the Space Station Systems include 12 types of local area networks with many variations of each type. In this context, the requirements for Space Station Systems state that an integrated approach is necessary in providing an end-to-end information system. More simply, the Station's information system must permit any user located at any local area network at any cluster to obtain information regardless of where it is stored. Thus a list of interface levels for users of the system must be defined. At the simplest level, there must be four transparencies to the user: it should be transparent to the user where the information he desires is located; it should be transparent to the user what kind of information replication is required; it should be transparent to the user how many processes are working on his behalf; and the fault tolerance aspects of the system should be transparent to the user.

Dr. McKay stated that the APSE has a tremendous potential for integrating various network models in order to meet Space Station System requirements. NASA/JSC is planning to develop the runtime support environments, the network operating system, the network configuration management, information management and communication management using Ada with a minimal amount of assembly code.

1.4 Integrated Support Software System

Mr. Raymond Szymanski from the Air Force Avionics Laboratory gave a presentation on the Integrated Support Software System (ISSS). The initial objective of ISSS was to provide the Avionics Simulation and Integration Laboratory (AVSAIL) and PAVE PILLAR a support software environment for JOVIAL and MIL-STD 1750A. The basic approach was to integrate Air Force owned tools with commercially available tools and tools developed in-house.

ISSS was developed to address the complete software life cycle. It was originally intended to be a laboratory environment only. Many off the shelf tools were incorporated within ISSS. ISSS is hosted on the VAX 11/780 and utilizes all the DEC provided utilities. To complement that, the IS/1 Workbench was also utilized. IS/1 Workbench has the same file system as the VAX. It has a word processing system, a screen oriented editor, a function key oriented terminal, and a number of other programming tools. One such tool is a configuration management tool called "MAKE". Once a makefile has been correctly defined, the user can rebuild his product by merely executing the makefile. Another tool, Source Code Control System (SCCS), allows the user to rebuild any product as it existed during any state of development.

Other tools being considered for future incorporation into ISSS include "USE.IT". Basically, it's a graphics building block approach to generating lines of high-order language code. Currently, USE.IT has proven to be an excellent tool for defining requirements and expections are high that once an avionics library is built, USE.IT may be able to generate high-order language code efficient enough for avionics applications.

Another program being examined is Software Design and Documentation Language (SDDL). SDDL permits a user to describe system requirements in an English-like format and the SDDL processes generate pseudo code. Thus far, this tool has successfully been used to "reverse engineer" software (i.e., provide documentation in the form of interface definitions, and flow diagrams for source code).

Current work on ISSS is focused on supporting the JOVIAL/1750A environment. Another tool, LODAYK has been developed to download VAX generated 1750A code to a 1750A processor.

Additionally, work is progressing on an effort to network a VAX 11/780 to a HARRIS 800 to provide simulation capabilities at the Laboratory. Current capabilities include virtual terminal from VAX to Harris and Vice versa.

Even though ISSS was developed as a Laboratory tool, it has been installed at two Air Logistics Centers at their request for the purpose of evaluating the environment for future ALC use. To date, the reaction to ISSS from the ALCs has been very positive with much interest being shown in the area of VAX/Harris networking.

1.5 Working Group Reports

Each E&V Team Working Group Chairperson presented a brief report concerning the activities scheduled during the individual Working Group sessions.

1.5.1 Requirements Working Group (REQWG)

The RECWG Chairperson, Major Dan Burton and Team member Lt John Prentice, can no longer participate with the E&V effort. Dr. Betsy Bailey and Dr. Tim Lindquist have been named Co-Chairpersons for this working group. The RECWG was to have delivered a draft Requirements Document this quarter, but because of the personnel turnover the draft document delivery will be a few weeks late.

1.5.2 Technical Coordination Working Group (TECWG)

The TECWG Chairperson, Jimmy Williamson, distributed the draft Technical Coordination Strategy document to the Team for review and comment by 22 June. The final version is scheduled to be available to the public on 6 July. The objective of this document is to present the strategy for identifying other E&V related efforts.

1.5.3 APSE Analysts Working Group (APSEWG)

The APSEWG Chairperson, Liz Kean, discussed the Group's accomplishments. Thus far, the APSEWG has identified five APSEs: the Ada Language System (ALS), the ALS/NAVY (ALS/N), the Ada Integrated Environment (AIE), Telesoft, and ROLM. A taxonomy for each of these environments has been developed. Future work includes a detailed comparison of these environments and publication of the results in an APSE Analysts Document.

1.5.4 CAIS Working Group (CAISWG)

The CAISWG Chairperson, Nelson Estes, discussed the Group's approach to developing the Validation Procedures Document. The draft document will reflect general validation procedures common to existing validation organizations. Later versions of this document will include CAIS specific validation procedures.

1.5.5 Public Coordination Working Group (PUBWG)

The PUBWG Chairperson Chris Anderson, distributed the minutes from the last E&V meeting and the Public Coordination strategy Document. Both documents are available to the public on the <EV-INFORMATION> directory, password "EV".

The PUBWG's plans for the working sessions during this meeting were presented and include: developing an E&V Status Report to be distributed at upcoming major conferences; and developing a Recommendations Questionaire based on the E&V Workshop Recommendations for distribution at the E&V Birds-of-a-Feather (BOF) Session at the AdaTEC meeting in July.

The general session of the E&V meeting was adjourned so that working groups could meet separately. Working groups met separately through Thursday.

2.0 Friday, 8 June, 1984

2.1 Working Group Status Reports

2.1.1 TECWG Status Report

The TECWG Status Report was given by Jimmy Williamson the Chairperson. No personnel changes were noted. The draft Technical Coordination Strategy Document was delivered this quarter for review by the Team. Also, the TECWG distributed a matrix listing E&V Team members who are also active in other potentially related E&V efforts. The only deliverable for next quarter is the final Technical Coordination Strategy Document.

2.1.2 PUBWG Status Report

The PUBWG Status Report was given by Chris Anderson, the Chairperson. No personnel changes were noted. Deliverables this quarter include the Public Coordination Strategy Document, the minutes from the last E&V meeting, the condensed version of the minutes for executive review, and the Project Reference List.

The PUBWG has requested the "Ada Fair" and "World of Ada" organizations to send the benchmarks to the E&V Chairperson. Current work includes the E&V Status Report to be distributed at upcoming conferences, the minutes of the E&V meeting, and the Recommendations Questionaire and accompanying viewgraphs for the AdaTEC BOF session in July.

2.1.3 APSEWG Status Report

The APSEWG Status Report was given by Liz Kean, the Chairperson. Personnel changes included Lt Rick Long's departure and the addition of Mars Gralia. Liz Kean is assuming the role of Chairperson in place of Rick Long. No deliverables were due this quarter. Accomplishments include the initial taxonomies of the AIE, ALS, ALS/N and Telesoft environments. The initial version of the APSE Analyst Report will be delivered at the September meeting for review and comments, with the final version delivered 28 September. An accompanying presentation will also be given. A strawman outline of the APSE Analyst Report format was presented. The outline includes five sections: Introduction, Scope, Identification of APSEs, Analyzed and Rationale, Description of Each APSE, and Comparison of APSEs. It was suggested and agreed upon to include an Executive Summary at the beginning of the Report.

2.1.4 CAISWG Status Report

The CAISWG Status Report was given by Nelson Estes, the Chairperson. No personnel changes were noted. No deliverables were due this quarter. The draft version of the Validation Policies and Procedures Document will be distributed to the Team for review in August. Work was initiated on the Test Requirements Document. This document consists of guidelines that a contractor will use to develop the CAIS validation tests. Presentations planned for the next meeting include a briefing concerning the draft Policies and Procedures Document.

2.1.5 REQWG Status Report

The REQWG Status Report was given by Dr. Betsy Bailey, the Co-Chairperson. Personnel losses include Dan Burton and John Prentice. New Members include Mike Burlakoff and Marlene Hazle. Dr. Betsy Bailey and Dr. Tim Lindquist will act as Co-Chairs for the group. Deliverables this quarter included the Requirements Document. Due to the personnel turnover, only a preliminary draft of this Document has been completed. An initial version of the Requirements Document will be sent to the Team for review in mid-August. The major sections of the Requirements Document will include: Introduction, Approach, APSE, E&V Requirements (based on components and attributes matrix) and Product Quality Guidance.

2.2 General Discussion

2.2.1 Active Items

Jinny Castor listed some administrative action items that she will handle. Team action items based on the Group Status Reports were also noted.

2.2.2 Agenda For September E&V Team Meeting

The distinguished reviewers (i.e., the E&V Workshop participants) will be invited to attend the first day of the Team meeting in September. It was decided to have a joint (distinguished reviewers/E&V Team) session for the first day of the meeting consisting of two panel sessions moderated by the CAISWG and REQWG. The CAISWG will present the CAIS Validation Policies and Procedures Document followed by an open floor discussion so that the distinguished reviewers can comment.

The CAISWG will also solicit input from the distinguished reviewers regarding the Test Requirements Document.

Similarly, the REQWG will present the Requirements Document followed by comments and discussion from the audience.

The second day of the meeting will feature presentations from the APSEWG, and Capt Rick Contreras, Hq AFOTEC, followed by separate working group meetings.

2.2.3 E&V Briefings

Jinny Castor encouraged the Team to present E&V briefings at conferences, Government meetings and at their home organizations. An action item was also given to the PUBWG to send the E&V Status Report to the publications identified in the Public Coordination Document as well as others such as the Intellimac and Sperry newsletters. An accompanying letter must also be drafted by the PUBWG for Jinny's signature.

2.2.4 Team Net Communication

In order to tag messages for comment by the Team or by particular working groups, various prefixes to message subjects were recommended. These prefixes include: TEAM/PC: (message for Team, Provide Comments); and Individual Working Group Name/PC (specific Working Group, provide Comments, example APSEWG/PC: subject); Other prefixes for informational messages only include: TEAM:subject (informational message for Team); and Individual Working Group Name (informational message for specific Working Group).

All messages should be copied to the entire Team.

The E&V meeting was then adjourned.

ATTENDANCE LIST
E&V Team Meeting, 6-8 June 1984

Virginia Castor
AFWAL/AAAF
Wright-Patterson AFB, OH

Chris Anderson
AFATL/DLMM
Eglin AFB, FL

Patrick Maher
MMECF
Hill AFB, UT

Mars Gralia
Johns Hopkins University
Applied Physics Laboratory
Laurel, MD

Don Jennings
OC-ALC/MMECE
Tinker AFB, OK

Jimmy Williamson
AFWAL/AAAF-2
Wright-Patterson AFB, OH

Marlene Hazle
MITRE Corporation
Bedford, MA

Lt Lori Mackey
CCSO/SKXX
Tinker AFB, OK

Lt Col Dick Cline
HQ AFOTEC/LG5
Kirtland AFB, NM

Capt Rick Contreras
HQ AFOTEC/LG55
Kirtland AFB, NM

Major James R. Johnson
AFWAL/AAAF-2
Wright-Patterson AFB, OH

Elizabeth Kean
RADC/COES
Griffiss AFB, NY

Lt Darleen Sobota
AFWAL/FIGR
Wright-Patterson AFB, OH

Capt John Taylor
HQ AFLC/MMEC
Wright-Patterson AFB, OH

Guy Taylor
FCDSSA
Virginia Beach, VA

Mike Burlakoff
Southwest Missouri State Univ
Springfield, MO

Richard Fleming
Aerospace Corporation
Los Angeles, CA

Charles McKay
University of Houston
Houston, TX

Mark Mears
AFWAL/FIGL
Wright-Patterson AFB, OH

Nelson Estes
ASD/AXTS
Wright-Patterson AFB, OH

Greg Bettice
Naval Avionics Center
Indianapolis, IN

R. J. Martin
Georgia Institute of Technology
Atlanta, GA

Elizabeth Bailey
Institute for Defense Analyses
Alexandria, VA

Jack Kramer
Institute for Defense Analyses
Alexandria, VA

Tim Lindquist
Virginia Polytechnic Institute
Blacksburg, VA

Gina Burt
AFALC/PTEC
Wright-Patterson AFB, OH

Raymond Szymanski
AFWAL/AAAF-2
Wright-Patterson AFB, OH

APPENDIX I

MINUTES

of the

EVALUATION & VALIDATION (E&V) MEETING

5-7 September 1984

Wright-Patterson Air Force Base, Ohio

Table of Contents

1.0 Wednesday, 5 September 1984

1.1 Welcome and General Business

The E&V meeting began with a welcome by the Chairperson, Jinny Castor, followed by self-introductions of Team members and Distinguished Reviewers (i.e., the E&V Workshop participants.)

Inputs were solicited regarding updating the E&V Plan, publication of the E&V Public Report, modification of Working Group responsibilities and Distinguished Reviewer participation in future E&V activities. Team members were also reminded to provide the necessary information to the PUBWG regarding reference material so that the Project Reference List can be updated.

The sources sought synopsis for the CAIS Validation Capability appeared in the Commerce Business Daily on 23 August. Unfortunately, the published synopsis did not appear as written. Consequently, a corrected publication will occur shortly.

The RFP for the E&V Technical Support effort will be issued by mid-September.

The Technical Coordination Strategy Document will be sent to program managers listed in the document for review (e.g., the JSSEE, WIS and STARS program managers.)

No further mass reproduction and mailing of documents will be performed for the E&V Team. Documents will be sent to interested individuals for review or will be available on the MILNET.

MILNET communication by Team members was encouraged.

The Deliverables Schedule was reviewed. To date, the following items have been delivered: The E&V Plan, the initiation of the Project Reference List, the Public Coordination Strategy Document, and Technical Coordination Strategy Document. Completion of the Requirements Document has slipped but should be completed by the end of September.

On August 1, 1984 Jinny Castor provided an E&V presentation at the SIGAda meeting in Hyannis, Ma. Over 200 copies of the E&V Recommendations Questionnaire and the Compiler Questionnaire were distributed at that meeting. An extremely low number have been completed and returned. Thus, it would appear that an alternate means for soliciting coordination must be found.

1.2 APSE Validation Procedures Document Review

Nelson Estes, the CAISWG Chairperson, presented an overview of the draft APSE Validation Procedures Document and solicited comments from the audience. A list of assumptions made by the CAISWG in developing the document was distributed to the team and discussed. One of the primary assumptions is that the CAIS 1.3 will not apply to embedded systems.

The draft document also implied that all APSE standards will be included in the CAIS. It was pointed out that this may be confusing since these other standards have not yet been developed. Eventually this document will address validation of all APSE components which have an associated military standard. Thus far, only the CAIS and the compiler fall into this category.

It was recommended that the proposed APSE Validation Organization be abbreviated as APSE_VO to distinguish it from the Ada Validation Organization (AVO). Also it was recommended that the E&V standard definitions of terms be used.

It was pointed out that this document will address validation of the implementation of the CAIS itself, not validation of tool interfaces to the CAIS. Whether or not a tool properly interfaces to the CAIS is strictly an evaluation issue unless the tool is addressed in a mandatory standard. In the latter case, the tool's interface may be subject to validation procedures. Written comments were collected by Nelson to be incorporated in the next version of the draft document.

1.3 E&V Requirements Document Review

Dr Tim Lindquist introduced members of the RECWG who then proceeded to discuss various portions of the Requirements Document. Tim pointed out that the document is still in draft form. He solicited critical review of those requirements already presented in the document as well as suggestions for additional requirements.

1.3.1 Introduction Section

Rich Fleming presented an overview of the Introduction Section of the document. He stated that the purpose of the document is to set forth requirements pertinent to the E&V effort. There are two primary types of requirements: those related to the E&V products for tools, methodologies, database, documentation, and E&V techniques; and those for the evaluation/validations for APSE components. Several review comments were offered.

It was pointed out that the scope of the E&V Task is to define the functionality of individual APSE components, not to define what constitutes an APSE.

It was recommended that the underlying assumptions be presented in the Introduction so that readers will be well informed as to the intent of the document.

1.3.2 General Requirements and Criteria for the E&V Methodology

Dr Elizabeth (Betsy) Kruesi Bailey presented an overview of Section 2 of the Requirements Document. Section 2 addresses the requirements on the E&V Team and on the E&V methodology. The document does not address requirements on the application of E&V technology.

1.3.3 Approach

Next, Betsy presented an overview of Section 3 of the Requirements Document which focuses on the approach of addressing the requirements listed in the document. The E&V Team Working Groups are the primary means of addressing the requirements on the E&V Team. The primary means of addressing requirements on technology development will focus on providing useful and usable information on APSE tools. This will be accomplished by developing the classification scheme (i.e., component/attribute matrix) for each tool to be evaluated and filling in the cells of the matrix with appropriate questions to be answered by E&V.

1.3.4 Required APSE Evaluation and Validations

Dr Tim Lindquist presented the introduction of Section 4, Required APSE Evaluations and Validations. He also outlined the contents of Section 4: attribute definitions, required component evaluations, and required macroscopic evaluations.

1.3.4.1 Attribute Definitions

Tim discussed the attributes that the APSE components are to possess. A lengthy discussion followed concerning various definitions.

1.3.4.2 Required Component Evaluations

This section is broken down into subsections, each addressing a different APSE component. The subsection first consists of the following items: a hierarchical breakdown of the component and accompanying definitions; component's hierarchical element followed by an attribute and one or more questions addressing the component's hierarchical element/attribute pair.

The components discussed thus far in the draft document include the Configuration Manager, Command Language Interpreter, and Compiler. Marlene Hazle discussed the Configuration Manager. It was also suggested that the interfaces of the various tools as they relate to each other (e.g., compiler, linker, run time system, etc.) are macroscopic issues that must also be addressed in the Requirements Document.

1.3.4.3 Required Macroscopic Evaluations

Tim Lindquist presented an overview of Section 4.4 of the Requirements Document dealing with macroscopic issues. These issues focus on the degree to which an APSE supports a particular software development methodology; tool support of life cycle phases; tool support of classes of mission-critical applications; and inter-tool interfaces.

The E&V meeting was adjourned for the day.

2.0 Thursday, 6 September 1984

The E&V meeting was opened by Jinny Castor. She outlined the agenda for the day and introduced the first speaker.

2.1 Air Force Operational Test and Evaluation Center (AFOTEC) Activities

Captain Rick Contreras from AFOTEC gave a presentation on AFOTEC's activities. AFOTEC is charged with evaluating embedded computer system software during the operational phase of the software life cycle. DoD Directive 5000.3 (26 Dec 76) provides the direction for AFOTEC activities. AFR 80-14 (12 Sep 80) further clarifies the methodology and AFOTEC's existence.

Software evaluation can be divided into two main areas: operational effectiveness and operational suitability. Automated support for evaluating operational effectiveness is nonexistent. Instead, close work with operating and developing agencies during the various life cycle phases of the software provides the necessary information for an evaluation of operational effectiveness. Automated support and methodologies are available for evaluation of operational suitability. The remainder of Rick's discussion was devoted to these support tools and methodologies.

Software supportability can be divided into two main areas: maintainability design considerations and software support resources. Rick defined software maintainability as the ability to provide a timely response to operational requirements. Several characteristics of software are examined: modularity, descriptiveness, consistency, simplicity, expandability, and instrumentation. At least ten aspects of each characteristic are examined via questions during an evaluation.

The software maintainability evaluation approach consists of documentation and source code listings examination followed by standard questionnaire completion. Evaluators (software experts) come from AFOTEC, the eventual support agency, and/or the using Command. The questionnaires feature a grading system for software characteristics. A numerical score is given. All questionnaires are publicly available through NTIS.

2.2 Independent Testing of an Ada Compiler

Dick Drake from IBM Federal Systems Division presented an overview of a project focused on independent testing of an Ada compiler in support of the Submarine Advanced Combat System (SUBACS) Program. The SUBACS Ada compiler was to be hosted on an IBM 370 and targeted to an M68000 and AN/UYK-44. In August 1983, IBM contracted Telesoft to produce the Ada compiler. The goal was to be able to use some amount of Ada in SUBACS (the remainder being CMS-2, Pascal and assembly language). The outcome is still uncertain at this point, with Ada compiler testing still continuing.

IBM's testing strategy for the Telesoft compiler concentrates on the functionality to be used on SUBACS. The compiler is a subset of Ada (i.e., no tasking, exceptions, or representation specifications). The SUBACS Program Design Language (PDL) is based on Ada.

There are several types of tests which IBM developed for the compiler: functionality tests and performance tests. Functionality tests include those developed by IBM; those developed by BITE, Inc; and those developed for Telesoft by Tachyon.

The performance tests were developed by OC Systems, Computer Science Corporation (CSC), and BITE, Inc. Some tests developed by OC Systems are based on the Ada Europe compiler selection guidelines. The code is not executed but manually inspected. Other performance tests developed by OC Systems consist of benchmark tests comparing Ada, Pascal, CMS-2, and Assembly Language. The benckmarks include a heapsort, bubble sort, and fixed-point tests. BITE, Inc. is currently developing capacity tests. Details of these tests are not yet available. CSC developed three types of benchmark tests: application, optimization detection, and language features. All tests are well documented.

IBMs heartiest emphasis has been placed on application specific testing. The primary approach is to develop prototype SUBACS applications concentrating on interfaces (i.e., language, tool, and run-time) and language features (i.e., sufficient subset).

Compiler ability will then be tested but these tests have not yet been developed.

Another interesting approach to testing that is being pursued is random test generation. The random generator works from templates of Ada and Pascal code. Two programs will be generated in Ada and Pascal so that a comparison can be made.

2.3 Working Group Meetings

The general session of the E&V meeting was recessed for a few hours so that the Working Groups could meet separately. The Distinguished Reviewers also met separately with Jinny Castor and LCDR Brian Schaar.

The general session reconvened.

The Distinguished Reviewers' meeting resulted in the following guidelines for future participation. The Distinguished Reviewers (or alternate) who attended this meeting (15 out of the original 21) are now permanent Distinguished Reviewers. The remaining six will no longer be considered Distinguished Reviewers. As long as the Reviewers participate, they will continue to be members. Each year new members may be added through participation in the annual E&V Workshop. The Distinguished Reviewers were distributed among the E&V Working Groups and are now members of those groups. Distinguished Reviewers are not eligible to be a working group chair.

3.0 Friday, 7 September 1984

3.1 Working Group Status Reports

3.1.1 TECWG Status Report

The TECWG Status Report was given by Jimmy Williamson, the Chairperson. Several personnel changes were noted: E&V membership - Randal Leavitt, replacing Kevin Chadwick; Distinguished Reviewers - Paul Dobbs and James Parlier, General Dynamics. Version 1.0 of the Technical Coordination Strategy Document has been finalized. No unresolved problems were reported. No deliverables are due next quarter.

3.1.2 PUBWG Status Report

The PUBWG Status Report was given by Chris Anderson, the Chairperson. One new member of the group was reported, Betty Wills. The minutes from the last meeting were distributed. The Quarterly Status Report was distributed at the SIGAda and AdaJUG meetings. Inputs to the Project Reference List were solicited from the E&V Team in order to maintain a current listing of our documents. The E&V Questionnaire based on the workshop recommendations was also distributed at the SIGAda meeting. Thus far, few responses have been received. Current work includes the E&V Status Report, the Minutes of the E&V meeting, Project Reference List updating, assistance in the publication of the Annual E&V Report, and the report from the Recommendations Questionaire.

3.1.3 APSEWG Status Report

The APSEWG Status Report was given by Liz Kean, the Chairperson. The working group name was changed from APSE Analysts to APSE Working Group. Six new members were noted: E&V members - Captain Albert Deese, Alternate; Doug Yarborough, GTE Government Systems; William Grabowski, GTE; Distinguished Reviewers - Paul Reilly, Data General Corp; Bard Crawford, TASC; and Marlow Henne, Harris Corp.

Commercial APSEs will not be described in the APSE Analyst Report due to potential legal problems. Thus, only DoD-developed APSEs will be described: the AIE, ALS, and ALS/N. Commercial APSEs will be examined for functionality input only. Version 1.0 of the APSE Analyst Report will be delivered by 30 September. No deliverables are due next quarter.

3.1.4 CAISWG Status Report

The CAISWG Status Report was given by Nelson Estes, the Chairperson. A new Chairperson was chosen, Darlene Sobota. Distinguished Reviewers include: Charles Hammons, Texas Instruments; John Reddan, SYSCON Corp., and Gary McKee, Martin Marietta. The E&V Policies Document will be expanded to include recommendations from the E&V Workshop. This document will be prepared by the CAISWG, reviewed by the E&V Team, and presented to Dr. Mathis for review and comment. Projected work for the next quarter includes a draft of the Assumptions Document and a redraft of the E&V Procedures Document, including ISO terminology.

3.1.5 REQWG Status Report

The REQWG Status Report was given by Betsy Baily, the Co-Chairperson. Four Distinguished Reviewers were added to the group: Helen Romanowski, Rockwell International; Raymond Sandborgh, Sperry Corp.; Mike Meirink, Sperry Corp., and Robert Fritz, CSC. Deliverables this quarter included Version 1.0 of the E&V Requirements Document which should be completed by 30 September. A lengthy discussion took place regarding slipped schedules versus publication of incomplete but frequently updated documents. It was generally felt that since the E&V documents are living documents, properly caveated "draft" versions are preferred to overdue polished versions.

Version 1.0 will include a list of assumptions. An attempt will be made to define the components in a MAPSE. After 1 October, the section on Product Quality Guidance will be developed. No deliverables are due next quarter.

3.2 General Discussion

The team meeting dates for the next year were listed. Updates to the E&V Plan as a result of this meeting were noted by Jinny. The E&V delivery schedule was reviewed and modified to reflect actual versus planned delivery dates. There was a lengthy discussion over the possibility of moving the meeting dates up farther from the end of fiscal quarters. This would permit more time to ready quarterly documents for delivery. This issue was not resolved. No more document reproduction will be provided to the Team. Most documents will be available on the net or in hard copy to volunteer reviewers. Recommendations for speakers at the December meeting were made by Team members.

A recommendation was made to include more rationale behind major decisions briefed in the Working Group Status Report. The Team and Reviewers agreed that this would be helpful.

A recommendation was also made for a separate meeting of the Reviewers to discuss and consolidate their observations regarding the E&V effort. Jinny suggested that the mechanisms for implementing this recommendation be formalized and presented over the net for review. Ray Sandburgh of Sperry Corp is responsible for this action item.

The E&V meeting was adjourned.

ATTENDANCE LIST
E&V Team Meeting, 5-7 September 1984

Jinny Castor
AFWAL/AAAF
Wright-Patterson AFB OH 45433

Patrick Maher
OO-ALC/MMECF
Hill AFB UT 84056-5609

Don Jennings
OC-ALC/MECE
Tinker AFB OK 73145-5990

Marlene Hazle
MITRE Corp
Bedford MA 01730

Capt Albert Deese
ASD/ADOL
Wright-Patterson AFB OH 45433

Major James R. Johnson
AFWAL/AAAF-2
Wright-Patterson AFB OH 45433

Guy Taylor
FCDSSA
Virginia Beach VA 23461

Richard Fleming
Aerospace Corp
Los Angeles CA 90009

Mark Mears
AFWAL/FIGLB
Wright-Patterson AFB OH 45433

Greg Bettice
Naval Avionics Center
Indianapolis IN 46218

LCDR Brian Schaar
Ada Joint Program Office
Room 3d 139 (400 A/N Dr)
The Pentagon
Washington DC 20301

Chris Anderson
AFATL/DLMM
Eglin AFB FL 32542

Mars Gralia
Johns Hopkins University
Applied Physics Laboratory
Laurel MD 20707

Jimmy Williamson
AFWAL/AAAF-2
Wright-Patterson AFB OH 45433

John Miller
SM-ALC/MME(1)D
McClellan AFB CA 95652

Capt Rick Contreras
HQ AFOTEC/LG55
Kirtland AFB NM 87117

Elizabeth Kean
RADC/COES
Griffiss AFB NY 13441

Douglas Olson, 2Lt
HQ AFCMD/SID
Kirtland AFB NM 87117

Betty Wills
CCSO/SKXX
Tinker AFB OK 73145-5990

Nelson Estes
ASD/AXTS
Wright-Patterson AFB OH 45433

Gary McKee
Martin Marietta
M/S 0423, P.O. Box 179
Denver CO 80201

Jack Kramer
Institute for Defense Analyses
1801 N. Beauregard St
Alexandria VA 22311

Gina Burt
AFALC/PTEC
Wright-Patterson AFB OH 45433

Marlow Henne
Harris Corp
GISD
150 Wickham Rd
Malbourne FL 32901

Terry D. Humphrey
NASA - Johnson Space Ctr
Mail Station EH-4
Houston TX 77058

John Reddan
SYSCON Corp
3990 Sherman St
San Diego CA 92110

Charles (Bud) Hammons
Texas Instruments
McKinney TX 75069

Helen Romanowsky
Rockwell International
Cedar Rapids IA 52498

Darleen Sobota, 1Lt
AFWAL/FIGR
Wright-Patterson AFB OH 45433

Bill Grabowski
GTE Gov't Systems
1 Federal St.
Billerica MA 01821

Paul Reilly
Data General Corp
4400 Computer Dr.
Westboro MA 01580

Amy Morganstern
EG&G, WASCI
8809 Sudley Rd
Manassas VA 22110

Randal Leavitt
PRIOR Data Sci./P.N.D. Canada
39 Highway 7
Nepean, Ontario
K2H 8R6

James F. Parlier
General Dynamics
DSD/WC
P.O. Box 85808, MZ VP 5300
San Diego CA 92139

Bard Crawford
TASC
1 Jacob Way
Reading MA 01867

Tim Lindquist
VPI
562 McBryde Hall
Blacksburg VA 24061

Elizabeth Kruesi Bailey
IDA
1801 N. Beauregard St
Alexandria VA 22311

Doug Yarborough
GTE Gov't Systems
1 Federal St
Billerica MA 01821

Bob Fritz
CSC
4045 Hancock St
San Diego CA 92110

Raymond E. Sandborgh
Sperry Corp
CS/DSD
P.O. Box 64525
St Paul MN

Ronnie Martin
Georgia Institute of Technology
School of Information & Comp Sci
Atlanta GA 30332

Mike Meirink
Sperry Corp
CS/DSD
P.O. Box 64525
St Paul MN

Paul Dobbs
General Dynamics
Box 748, MZ 54044
Ft Worth TX 76114

Georgeanne Chitwood
ASD/ADOL
Wright-Patterson AFB OH 45433

Dick Drake
IBM/FSD
MANASSAS VA 22110

APPENDIX J

EVALUATION CRITERIA FOR ADA COMPILERS

11 September 1984

Table of Contents

EVALUATION CRITERIA FOR ADA COMPILERS

11 September 1984

Author: Elizabeth Kean (Rome Air Development Center) in support of the E&V Team

In June 1983, the AJPO proposed the formation of the E&V (Evaluation and Validation) Task and a tri-service APSE E&V Team, with the Air Force (Air Force Wright Aeronautical Laboratories (AFWAL)) designated as the lead service. The purpose of the E&V Team is to develop the techniques and tools which will provide a capability to perform assessment of APSEs and to determine the conformance of APSEs to the CAIS (Common APSE Interface Set).

In order to accomplish this goal, a criteria questionnaire was developed and distributed at the SigAda meeting (30 July - 1 August 1984) in Hyannis, MA. Based upon comments received regarding this questionnaire, the following list of evaluation criteria for Ada compilers was composed.

The criteria/tests are designed to complement the Ada Compiler Validation Capability (ACVC) test suite. The criteria are collected according to the set of attributes defined in the Requirements for Evaluation and Validation of Ada Programming Support Environments, dated 17 October 1984. Compiler areas yet to be covered indepth include the run-time system, program library tools, linking and loading facilities, etc.

AVAILABILITY

1. Have there been any major software systems developed using this compiler?

2. For each of the compiler's targets, approximately how many users are there? To what degree have the targets been exercised?

3. Can the compiler be invoked in both an interactive and a batch mode?

4. Can the compiler be invoked while using other APSE tools? (While in the editor, for example.)

5. Are there any additional tools supplied with the compiler (e.g., symbolic debugger, target simulator, downloader, linker, etc.)?

6. Are there significant compiler features that could be considered above and beyond that specified in the Ada language specification?

CAPACITY

1. What is the maximum number of errors detectable on a single line?

2. What is the maximum number of errors detectable in a single compilation unit?

3. What is the maximum number of Ada statements allowed for a single compilation unit? How are statements to be counted for this purpose?

4. What is the maximum number of compilation units allowed for a single file?

5. What is the maximum number of symbols allowed per compilation unit?

6. What is the maximum length allowed for a source line?
KNOWN TESTS:
(AIE)fen0801a, (AIE)fen0801b

7. What is the maximum size of the Intermediate Language that can be generated?

8. What are the maximum number of elements allowed in an enumeration type?

9. What is the maximum level of nesting in packages, loops, cases, record type declarations, IF-statements, blocks, etc?
KNOWN TESTS:
Packages: (AIE)fen0807a, (AIE)fen0807b, (AIE)fen0807c
Loops: (AIE)fen0805a, (AIE)fen0805b, (AIE)fen0805c
IFs: (AIE)fen0806a, (AIE)fen0806b, (AIE)fen0806c
Blocks: (AIE)fen0804a, (AIE)fen0804b, (AIE)fen0804c
Subprograms: (AIE)fen0808a, (AIE)fen0808b, (AIE)fen0808c, (AIE)fen0809a

10. What are the maximum number of operands in expressions?

11. What is the maximum number of WITHed units allowed?
KNOWN TESTS:
196-199 WITHs: (AIE)fen0803a, (AIE)fen0803b, (AIE)fen0803c, (AIE)fen0803d, (AIE)fen0803e

12. What is the maximum number of DIANA structures (i.e., symbol table entries) imported for a single compilation?

13. What is the maximum number of tasks that can be in existence?

14. What is the maximum number of tasks that can be spawned for a single program?

15. What is the maximum length of an identifier?

16. What is the maximum number of discriminants in a constraint?

17. What is the maximum number of fields in a record aggregate?

18. What is the maximum number of formals in a generic unit?

19. What is the maximum number of nested contexts?

20. What is the maximum number of indices in an array aggregate?

21. What is the maximum number of parameters in a call?

22. What is the maximum depth of expansion of INLINE subprograms?

23. What is the maximum precision of the fixed point, floating point and integer arithmetic?
KNOWN TESTS:
(AdaFair84)cauchfl.ada, (AdaFair84)cauchfy.ada,
(AdaFair84)univ_ar.ada, (AdaFair84)cauchun.ada, (AVO)MATHTEST

24. What is the maximum/minimum size of the run-time system with/without tasking?

25. What is the size of the each of the phases of the compiler?

26. What is the minimum memory required to execute the compiler?

27. Is LOW_LEVEL_IO supported?

28. Do SEQUENTIAL_IO and DIRECT_IO "Work as expected?" Under what conditions, if any is USE_ERROR raised?

29. Do the compiler and runtimes implement non-blocking I/O and non-blocking operating system service calls?

30. What is the range of PRIORITY? What is the runtime task scheduling algorithm?

31. Are any limitations imposed on the use of pragmas SYSTEM_NAME,

STORAGE_UNIT, and MEMORY_SIZE?

32. What is the maximum number of overloadings per identifier?

33. Can generic bodies be compiled separately from their specifications?

34. Can any instances of erroneous execution be detected?

35. Can deadlock be detected?

36. What overhead is involved in context switching between tasks?
KNOWN TESTS:
(SRI)chain2, (SRI)chain5, (SRI)chain10, (SRI)chain20

37. How does the Ada compiler perform when compiling various applications programs?
KNOWN TESTS:
(AdaFair84)bsearch.ada, (AdaFair84)random.ada, (AdaFair84)set.ada
(AVO.WORLDOF)directory_utility, (AVO)KPTOOLS, (AVO)FORMATTER,
(AVO)STUBBER

38. How large is the compiler?

39. Is the compiler ever completely in memory?

40. What is the value of each of the predefined attributes (e.g., INTEGER'FIRST, INTEGER'LAST, FLOAT'DIGITS, etc.)

41. What is the maximum number of users that can invoke the compiler simultaneously? Has this number been verified?

CONFIGURATION MANAGEMENT (CM)

1. What is the general CM plan?

2. Who is responsible for CM (for the supplier)?

3. Are all the source and object modules for a complete version available in one area (e.g., on a tape or a separately controlled disk area)? Is this area accessible to only one person or anyone on the project?

4. How are compiler fixes or enhancements incorporated into a new version?

5. How are new versions controlled and released? Which of these versions will be recertified or revalidated?

6. If the compiler produces code for more than one target, how are common and machine-dependent modules controlled? For example, are there conventions for naming common function modules for each target?

7. How are approved compiler changes incorporated into new versions? (The change approval procedures).

8. Does there exist a list of all modules (including the run time system) that are needed for a complete compiler version? (A version description document).

COST

1. What are the costs of acquiring the compiler? These costs should be given in terms of:

a. Does the cost include the installation and required maintenance support?

b. Is source code or only object code supplied?

c. Costs of additional (non-supplier developed tools)?

d. Monthly (or periodic) additional maintenance fees?

e. Does the cost include receipt of new versions of the compiler and needed tools?

f. Various cost options depending on licensing (proprietary) arrangements?

2. What is the estimated cost for a compiler rehost?

3. What is the estimated cost for a compiler retarget?

4. What do the costs for rehosting and retargetting include (test, integration, installation, etc.)?

DOCUMENTATION

1. Is a requirements document available? Verify the content and quality.

2. Are complete design specifications available? Verify the content and quality as follows:

a. An overview of the compiler design showing the major structure and design.

b. Details of the compiler phases and passes.

c. Separate sections which outline the design of the host and each target (to include the run time system(s)).

d. Is the design detail sufficient such that an experienced compiler software engineer could maintain the compiler?

e. Are compiler design changes updated in the documents?

3. Is sufficient user documentation available for the host and each target? Is it on-line?

4. Does the users (or reference) manual contain an Appendix F which describes all implementation dependent characteristics?

5. Is the compiler release and compilation date identified on the hard copy listing?

6. Is documentation available for any special tools that were used for the compiler development?

7. Is documentation available for any separate tools that are needed for compiler operation?

8. What are the procedures used to update the documentation as a result of compiler changes? Who is responsible to verify that this is done?

9. Are there any alternative structures suggested for the production of low cost low quality back ends and high quality back ends?

EFFICIENCY

(For the following set of questions, speed is the number of source statements per CPU minute.)

1. What is the speed of object code generation (from the beginning of compilation)?
KNOWN TESTS:
(AIE)ben0701a

2. What is the speed of the Front-End if it can be run separately?

3. What is the speed of the Back-End if it can be run separately?

4. What is the speed of listing generation?

5. What is the speed of compilation with OPTIMIZATION=time? OPTIMIZATION= space? OPTIMIZATION=none?

6. Do the pragmas supported by the compiler, if any, impact the compilation speed?

7. What target machine independent optimizations are performed? For example, does the compiler perform constant propagation, constant folding, common subexpression elimination, expression simplification, strength reduction, range and constraint checks, removing unreachable code, cross-jumping, task context switch minimization, code sharing between generic instantiations, etc.?
KNOWN TESTS:
(AIE)ben0301a, (AIE)ben0301b, (AIE)ben0302a, (AIE)ben0302b,
(AIE)ben0302c, (AIE)ben0302t, (AIE)ben0302e, (AIE)ben0303a,
(AIE)ben0303b, (AIE)ben0303c, (AIE)ben0303d, (AIE)ben0303e,
(AIE)ben0303f, (AIE)ben0303g, (AIE)ben0303h, (AIE)ben0303i,
(AIE)ben0303j, (AIE)ben0303k, (AIE)ben0303l, (AIE)ben0303m,
(AIE)ben0304a, (AIE)ben0304b, (AIE)ben0304c, (AIE)ben0304d,
(AIE)ben0304e, (AIE)ben0304f, (AIE)ben0304g, (AIE)ben0401a,

(AIE)ben0401b, (AIE)ben0402a, (AIE)ben0501a, (AIE)ben0501b, (AIE)ben0501c, (AIE)ben0501d, (AIE)ben0502a, (AIE)ben0502b, (AIE)ben0502c, (AIE)ben0502d, (AIE)ben0502e, (AIE)ben0502f, (AIE)ben0502g, (AIE)ben0502h, (AIE)ben0502i, (AIE)ben0503a, (AIE)ben0504a, (AIE)ben0504b, (AIE)ben0504c, (AIE)ben0504d, (AIE)ben0504e

8. What target dependent optimizations are employed?

9. What is the probable effect of optimization on the raising and handling of exceptions (ess LRM 11.6)?

10. To what extent does the user control the optimizations?

11. Can the user specify the use/non-use of specific optimizations?

12. How does using select alternatives affect the performance of the executable code?
KNOWN TESTS:
(SRI) guard2, (SRI) guard20, (SRI) guard20e, (SRI) guard20et, (SRI) guards20t, (SRI) guards2e

13. Do idle tasks impact the performance of the executable code?
KNOWN TESTS:
(SRI)idle1, (SRI)idle5, (SRI)idle10, (SRI)idle20

14. Is it better to have many small tasks with single entry choices or a few large tasks with many select choices?
KNOWN TESTS:
(SRI)moretasks, (SRI)moretasks1, (SRI)moreselct, (SRI)moreselctr

15. Does the ordering of entry clauses in a SELECT impact execution speed?
KNOWN TESTS:
(SRI)order31, (SRI)order31r, (SRI)order32, (SRI)order100

16. If multiprocessing is supported by the implementation, are Ada tasks mapped to a single underlying process, or is each task mapped to a separate process?

17. If the target is a distributed system, how is "immediately" defined for conditional entry calls?

18. Does the size of a parameter affect performance?
KNOWN TESTS:
(SRI)passarrys, (SRI)passarryb, (SRI)passinout

19. Does the number of select choices affect performance?
KNOWN TESTS:
(SRI)select2, (SRI)select2e, (SRI)select20, (SRI)select20e

20. Can the Ada scheduler starve a task?
KNOWN TESTS:
(SRI)schedtest

21. What is the size of the resulting object code for OPTIMIZE=none? OPTIMIZE=space? OPTIMIZE=time?

22. What is the CPU time required for execution of code compiled with OPTIMIZE=none? OPTIMIZE=time? OPTIMIZE=space?
KNOWN TESTS:
(AVO.WORLDOF)eratos.r, (AVO.WORLDOF)float_test.r,
(AVO.WORLDOF)generator.r, (AVO.WORLDOF)io_test.r,
(AVO.WORLDOF)item_body.r, (AVO.WORLDOF)item_spec.r,
(AVO.WORLDOF)lf_disk_80.tex, (AVO.WORLDOF)lf_pgmap.text,
(AVO.WORLDOF)list_body.r, (AVO.WORLDOF)sieve_task.r,
(AVO.WORLDOF)stinger.r, (AVO.WORLDOF)timing.r, (AVO.WORLDOF)tower.r,
(AVO.WORLDOF)tran_spec.r, (AVO.WORLDOF.juggling1)e_ball_1.r,
(AVO.WORLDOF.juggling1)l_hand_1.r, (AVO.WORLDOF.juggling1)o_ball_1.r,
(AVO.WORLDOF.juggling1)r_hand_1.r, (AVO.WORLDOF.juggling2)counter.r,
(AVO.WORLDOF.juggling2)even_ball.r, (AVO.WORLDOF.juggling2)get_jug.r,
(AVO.WORLDOF.juggling2)left_hand.r, (AVO.WORLDOF.juggling2)monitor.r
(AVO.WORLDOF.juggling2)odd_ball_1.r,
(AVO.WORLDOF.juggling2)print_pos.r,
(AVO.WORLDOF.juggling2)right_hand.r

23. What is the difference in size between object code generated by an Ada compiler as compared with other languages (e.g., Pascal, COBOL, C, FORTRAN, Assembler, etc.)?

24. What is the difference in execution time between object code generated by an Ada compiler as compared with other languages?

25. What is the execution time (in CPU seconds) for file operations (with TEXT_IO, DIRECT_IO, etc.)?
KNOWN TESTS:
(AdaFair84)char_dir.ada, (AdaFair84)char_pnum.ada,
(AdaFair84)int_dir.ada, (AdaFair84)char_text.ada,
(AdaFair84)int_text.ada

26. What is the execution time (in CPU seconds) for arithmetic and logical operations?
KNOWN TESTS:
(AdaFair84)boolvec.ada, (AdaFair84)floatvec.ada,
(AdaFair84)intvec.ada, (AVO)AHL, (AVO)AUSSIE, (AVO)BASEMATH, (AVO)ADD,
(AVO)MULT, (AVO)PUZZLE, (AVO)SEARCH, (AVO)SIEVE,
(AVO)SYNTHETIC(Whetstone), (AVO.MARK)ADD, (AVO.MARK)MULT,
(AVO.MARK)PUZZLE, (AVO.MARK)SEARCH, (AVO.MARK)SIEVE,
(AVO.MARK)SYNTHETIC(Whetstone)

27. What is the execution time (in CPU seconds) for simple procedures with scalar parameters?
KNOWN TESTS:
(AdaFair84)proccal.ada

28. What is the execution time (in CPU seconds) for a simple rendezvous?
KNOWN TESTS:
(AdaFair84)rendez.ada

29. What is the effect on code size and execution time of the pragma SUPPRESS?

30. What is the effect on code size and execution time of using

UNCHECKED_DEALLOCATION vs. pragma CONTROLLED vs. neither?

31. What is the effect on code size of a DEBUG option, if is exists?

32. What is the procedure calling overhead for procedures and functions?
KNOWN TESTS:
(AdaFair84)ackerman.ada, (AVO)ACKER, (AVO.WORLDOF)ackerman.r

33. Are there Ada features that produce more efficient object code with respect to time and space?
KNOWN TESTS:
(AdaFair84)qsortpar.ada, (AdaFair84)qsortseq.ada

EXTENDABILITY

1. What were the original design goals of the compiler?

a. Was the design intended for a particular class of users?

b. Are any specific applications envisaged?

2. Is the compiler is written in Ada, were the use of certain language constructs avoided (e.g., tasking, generics, real arithmetic)? If so, which ones and why?

3. If the compiler is written in Ada, has it successfully recompiled itself?

4. Were any special tools such as a compiler-compiler, translator writing systems, etc., used during the development. If so, are they available to possibly construct additional tools?

a. Do these tools generate a source program of the compiler or do they translate directly into object code?

b. If these tools do not generate an Ada (or other language) program, how can the tools be retargeted?

c. What languages are the tools written in?

GRANULARITY

1. What are the major compiler phases? What phases are in memory as the compilation progresses?

2. To what degree are the components of the compiler separately executable tools? Is their use documented?

3. What parts of the compiler are seen as useful in building other tools?

HARDWARE

1. What are the host/target pairs for the compiler?

1. For each target, does the compiler directly output (relocatable) object code?

3. For each target, does the compiler output assembly language? Is it a standard version?

4. What is the character set of the host? of the target?

5. Are hardware machine dependencies clearly identified in both the code and documentation?

6. Is the compiler designed to use virtual memory?

7. Is the compiler designed to take advantage of a multiprocessing implementation? (Was the compiler designed to use multitasking?)

8. Are hardware dependencies concealed by module interfaces?

9. Does the compiler support distributed machine targets? If so, which ones?

INTERFACES

1. Is the major design interface to a KAPSE or the host operating system?

2. If the interface is to a KAPSE, what KAPSE facilities does the compiler use?

3. Does the compiler operate in a particular APSE? If so, what APSE (or MAPSE) tools does it require, if any?

4. If not part of an APSE, what characteristics of the host operating system does the compiler rely on? Are all such system dependencies concealed behind module interfaces?

5. Which interfaces are regarded as significant for rehosting or retargeting?

6. What other tools (e.g., symbolic debugger) does the compiler interface with?

a. To what extent are the interfaces documented?

b. Can alternative tools be written conforming to these interfaces?

INTEROPERABILITY

1. What compiler generated information is available to other tools? Symbol table? Cross-reference table? Intermediate forms? Listing outputs?

2. Does the compiler share, or make use of, other APSE (or operating system) tables or information?

MAINTAINABILITY

1. What language(s) is the compiler written in?

2. Are instructions available to enable a non-compiler person to install the compiler on an identical host system?

3. Are the procedures for complete compiler generation (from source to executable) documented?

4. What arrangements are available for maintenance? Such arrangements can range from postal service to an on-call maintenance staff.

5. What is the quality of maintenance support?

    a. Designated persons for maintenance contract?

    b. Availability of maintenance documentation?

    c. Telephone query service, visits by supplier staff, courses, etc?

6. What are the arrangements for charging for maintenance and/or support of the compiler?

PROPRIETARY

1. Can a user install the compiler, or must the supplier do the installation?

2. Are there any proprietary restrictions on compiler release (e.g., no source supplied, data rights, etc.)?

3. Are there any restrictions on special (non-supplier developed) tools needed for compiler operation? Also, for any optional tools that may be useful?

4. Does the supplier allow others to perform a rehost or retarget?

5. Under what circumstances may the source be made available for a rehost or retarget?

6. What are the licensing arrangements for the compiler (e.g., at how many sites can the compiler be used)?

7. What agreement does the user have to sign before the compiler may be supplied to others?

8. Can a license to distribute the compiler to others be bought or leased? What parts of the compiler (run time system, packages,

separate tools, etc.) can be distributed? Can source be included?

9. Can a license to use the compiler be bought outright or leased?

10. What are the arrangements (if any) for the release of information about the compiler's internal structure?

11. Are there any restrictions on the use and/or distribution of software produced by the compiler? It should be noted that the software produced often contains a run time system delivered by the compiler supplier.

REHOSTABILITY

1. Has the compiler been rehosted?

2. What module (or modules) of the front end (machine independent) need to be modified for the rehost?

3. Is there a manual which describes the steps necessary to rehost the compiler?

4. Are system dependencies adequately isolated and documented?

5. Is there a kit of tools and/or components available to help with the rehosting task?

6. Is the compiler sufficiently modular to allow implementation of critical parts (such as major data structures) to be easily altered for the rehost task?

7. Is an estimate of time given for the rehost?

RETARGETABILITY

1. What modules of the back end (machine dependent) need to be modified for a retarget?

2. What modules of the front end need to be modified for a retarget and what are their interfaces?

a. What techniques are used: Package standard, parameter file, a special package that is linked in, etc.)? Hopefully, this is not wired in to the compiler.

3. Are there any automated tools to aid in the retarget process?

4. Is an estimate of time given for the retarget task?

5. Is there a manual describing the procedures for retargeting? Possibly with examples.

6. For the intermediate language retargeting interface, is the intermediate language tree structured, linear, etc?

7. Is there more than one level of intermediate language at which retargetting is carried out?

8. For the retarget process, what assumptions are made in the design of, and requirements of, the run time system (e.g., tasking monitor, storage allocation scheme, etc.)?

9. For a retarget, do the presence of other tools in the compilation system or APSE affect the back end?

ROBUSTNESS

1. What safeguards are implemented for protection and recovery against unforeseen system, user and its own failures? Data protection? Internal exception handlers? Trace back facility?

2. As more than one simultaneous user invokes the compiler, by how much does the mininum size of memory needed to run the compiler increase?

3. Are any resources other than primary and secondary storage needed to invoke the compiler?

4. Are compiler phases overlaid to reduce memory occupancy? If so, are any requirements placed on the system?

TEST AVAILABILITY

1. What tests are available from the supplier to verify compiler operations?

   a. Are the tests documented in a test plan?

   b. Are instructions for use available?

USABILITY

1. If the compiler interface is to the host operating system, under which operating system version (or release) does the compiler operate? Also, which target(s) operating system(s)?

2. If the compiler interface is to an APSE, under what version (or release) of the APSE does the compiler operate?

3. What are the values outside the range of safe numbers for real types?

4. Are there any restrictions on the use of the generic procedure UNCHECKED_DEALLOCATION?

5. Are there any restrictions on the use of the generic procedure UNCHECKED_CONVERSION?

6. Does the compiler generate a history file which records source

file name, compilation unit name, program library name, owner(s) of each and the date/time of the compilation?

7. Does the compiler record the versions of all compilation units used in the compilation?

8. How many passes over the code (source and IL) does the compiler make?

9 What compiler options are available? For example, can the user specify the following options: no semantics, no code, comments, lookahead, optimize (off, space, time, etc), choice of runtime kernels, compiler maintenance, compiler new version of package STANDARD, debug, main program identification, prettyprint source, provide user with traceback information for unhandled exceptions, etc?
KNOWN TESTS:
(AIE)fen0501a, (AIE)fen0502a, (AIE)fen0503a, (AIE)fen0503b,
(AIE)fen0503c, (AIE)fen0503d, (AIE)fen0503e, (AIE)fen0503f,
(AIE)fen0504a, (AIE)fen0504b, (AIE)fen0504c, (AIE)fen0504d,
(AIE)fen0504e, (AIE)fen0504f, (AIE)fen0505a, (AIE)fen0505b,
(AIE)fen0506a, (AIE)fen0506b, (AIE)fen0506c, (AIE)fen0506d,
(AIE)mid0501a, (AIE)mid0502b, (AIE)mid0502d, (AIE)mid0503a,
(AIE)ben0201a, (AIE)mid0501b, (AIE)mid0501c, (AIE)mid0502c,
(AIE)ben0102a (AIE)mid0502a, (AIE)ben0101a, (AIE)ben0202a

10. What control does the user have over listings? For example, can the user control: listing source text, reformatting the source, listing text of an instantiated generic unit, listing private parts of packages, listing attributes of all symbols in source text, listing cross reference information, listing statistics, listing machine/assembly code, listing the intermediate language generated, listing diagnostics, listing use of machine-dependent features, pragmas, etc.?

11. Does abnormal termination leave a consistent program library? If not, how can a user restore consistency? Are checkpointing and/or transaction updating supported?

12. Is the intermediate language generated available for other tools?

13. Can the source be reconstructed from the intermediate language (e.g., DIANA, or Abstract Syntax Tree)?

14. Are all LRM pragmas properly recognized or handled? What implementation dependent pragmas are supported, if any?
KNOWN TESTS:
Pragma OPTIMIZE: (AIE)mid0503a, (AIE)mid0503b
Pragma STATIC: (AIE)mid0504a, (AIE)mid0504b
Pragma INLINE: (AIE)mid0505a, (AIE)ben0505a, (AIE)ben0505b

15. Are the parameter passing methods and subprogram calls handled efficiently?
KNOWN TESTS:
PASSED BY VALUE (<= 64 BITS): (AIE)mid0701a, (AIE)mid0701b
PASSED BY VALUE (>= 64 BITS): (AIE)mid0702a, (AIE)mid0702b
SELECTED PASSING METHODS: (AIE)mid0703a, (AIE)mid0703b
PASSED BY REFERENCE (DYNAMIC): (AIE)mid0704a, (AIE)mid0704b

16. Are there any restrictions on unchecked conversions? on unchecked deallocation?
KNOWN TESTS:
(AdaFair84)lowlev.ada

17. Will the compiler generate code to automatically trace the execution of a statement, group of statements, or module?

18. Will the compiler generate code to track changes in value of any variable and print changes?

19. Will the compiler generate code to provide statistics on the execution time/frequency of usage of any segment of code?

20. Will the compiler generate code which will trace the occurrence of exceptions and the levels at which they are handled?

21. If representation specifications are implemented, are there any restrictions? If so, what are they?
KNOWN TESTS:
(AdaFair84)derived.ada

22. What is the accuracy of the error message positioning?

23. What is the time consumption for error detection and recovery?

24. What is the clarity of error messages in terms of the language?
KNOWN TESTS:
(AdaFair84)friend.ada

25. Will the compiler produce an executable object program even if errors are present? If so, under what conditions and what is the effect of executing a source statement that contains an error?

26. Are steps taken to avoid "cascading" of compilation errors?

27. What warning messages does the compiler print? For example, warning for ignored pragmas, warning for an unusually expensive construct, etc.? Is the user able to switch off certain warning messages?

28. Can the user control the level of error which will abort the compilation?

29. Can the user halt the compilation after some stage, examine the current state of the compiler output and restart the compiler?

30. Is the compiler re-entrant?

31. Does using the debugger, if one exists, require compiling the software with a special DEBUG option?

32. Can two or more compilations access and/or update the same library simultaneously? If so, how is consistency maintained?

33. Can any instances of erroneous execution be automatically

detected? How are such instances handled?

REFERENCES

1. Generic APSE Evaluation Questions, General Dynamics

2. Ada Integrated Environment (AIE) Test Procedures for the Compiler Front-End, Intermetrics, Inc. (Available through RADC upon completion of the AIE contract)

3. AIE Test Procedures for the Compiler Middle-Part, Intermetrics, Inc. (Available through RADC upon completion of the AIE contract)

4. AIE Test Procedures for the Compiler Back-End, Intermetrics, Inc. (Available through RADC upon completion of the AIE contract)

5. AIE Test Procedures for the Compiler Subsystem, Intermetrics, Inc. (Available through RADC upon completion of the AIE contract)

6. SRI Tests (Available on USC-ECLB<EV-INFORMATION>)

7. AdaFair 84 Tests (Available on USC-ECLB<EV-INFORMATION>)

8. World of Ada Tests (Available through AJPO)

9. Ada-Europe Guidelines for Ada Compiler specification and selection by J C D Nissen and B A Wichmann (et al) (ACM Ada Letters, July, August 1983)

10. Mike Burlakoff, University of Missouri

11. Requirements for Evaluation and Validation of Ada Programming Support Environments, Version 1.0, XX XXX XX.

**It is the intent of the E&V Team to eventually make as many tests as possible available through EV-INFORMATION.

ADA FAIR '84 TESTS

ackerman.ada	recursion, procedure calling overhead
boolvec.ada	time for "AND" operation on boolean vector
bsearch.ada	generic binary search
cauchfl.ada	floating point accuracy
cauchfy.ada	fixed point accuracy
univ_ar.ada	universal real and integer arithmetic
cauchun.ada	test universal arithmetic package
char_dir.ada	time for file operations using DIRECT_IO with characters
char_pnum.ada	time for file operations using TEXT_IO and ENUMERATION_IO with characters
char_text.ada	time for file operations using TEXT-IO with characters
int_dir.ada	time for file operations using DIRECT_IO with integers
int_text.ada	time for file operations using TEXT_IO with integers
conprod.ada	tasking performance using buffering task
derived.ada	inter_conversion of derived types with different representations
floatvec.ada	time for adding elements of a large floating point vector
intvec.ada	time to add elements to a large integer vector
friend.ada	friendliness of compiler - warnings,exceptions
lowlev.ada	test length of clauses and unchecked conversion
proccal.ada	time for simple procedure calls with scalar parameters
qsortpar.ada	compare parallel and sequential sort algoritnms
qsortseq.ada	compare parallel and sequential sort algorithms
random.ada	random number generation, generate linear congruential sequences
rendez.ada	time for simple rendezvous
set.ada	implementation of sets
shared.ada	tasking to provide shred access to global variables

AVO TESTS

ACKER	ackerman function
AHL	test arithmetic and random number
AUSSIE	machine arithmetic
BASEMATH	simulation of base number arithmetic for comparing
COMP	check assumptions about behavior of computer
COMPA	declarations - separate statements and lines
COMPB	declarations - one statement and many lines
COMPC	assignments - several per line
COMPD	assignments - one per line
COMPE	alternating comments and assignments - one per line
COMPF	comments followed by assignments - one per line
COMPG	tests linearity, alternating comments and assignments
COMPN	null procedure
COMPT	with and use
COMPZ	test program - 1 assignment 1 declaration
MARK	set of benchmarks
ADD	simple addition
MULT	simple multiplication
PUZZLE	move pieces
SEARCH	find character string
SIEVE	byte prime number benchmark
SYNTHETIC	synthetic benchmark
MATHLIB	elementary math functions
MATHTEST	test elementary math functions
KPTOOLS	insulate "software tools" from system dependencies
FORMATTER	pretty printer system - put code in military standard
STUBBER	creates package bodies from text
WORLDOF	set of thirty-nine benchmarks
ackerman.r	enumeration types, non-primitive recursion, speed requirements, space requirements
ambiguity.r	overloading, disambiguation, package linkage, separate compilation
dir_body.r	locate, insert, delete, create - item/dictionary
dir_spec.r	directory functions - create, delete, change, add, display, scan
dir_util.r	create and manipulate directory using directory functions
eratos.r	iterative statements, basic arrays, speed of execution
float_test.r	floating points with transcendental functions
generator.r	separate procedure specifications and body specifications
io_test.r	I/O capabilities
item_body.r	get, display item
item_spec.r	get, display, compare items
lf_disk_80.tex	disk functions
lf_pgmap.text	page functions
list_body.r	generics, private types, pragmas, exceptions
list_test.r	generics, overloading, private types, pragmas, exceptions
manager.r	generics, overloading, private types, pragmas, exceptions
overload.r	ambiguous overloading of "+"
resolution.r	overloading

```
  roman.r             overloading, disambiguation, with Roman_Numerals
                      package
  roman_body.r        overloading, disambiguation, with Roman_Numerals
                      package
  sieve_body.r        body of sieve task
  sieve_task.r        solve sieve of Erotothenes problems using tasking
  stinger.r           basic string operations
  timing.r            timing functions
  tower.r             records, discriminants, non-primitive recursion,
                      execution speed
  tran_body.r         body of transcendental functions using iteration
  tran_spec.r         number crunching abilities
juggling1.r           tasking, entry calls, exceptions, subunits,
                      single compilation of multiple units
  e_ball_1.r          tasking, subunits
  l_hand_1.r          tasking, subunits
  o_ball_1.r          tasking, subunits
  r_hand_1.r          tasking, subunits
juggling2.r           tasking, selective waits, families of entries,
                      exceptions, subunits, separate compilations
  counter.r           same as juggling2.r
  even_ball.r         same as juggling2.r
  get_jug.r           same as juggling2.r
  left_hand.r         same as juggling2.r
  monitor.r           same as juggling2.r
  odd_ball_1.r        same as juggling2.r
  print_pos.r         same as juggling2.r
  right_hand.r        same as juggling2.r
```

SRI TESTS

ALL TESTS ARE ADA TASKING TESTERS

```
DETERMINE OVERHEAD IN CONTEXTS SWITCHES BETWEEN TASKS
  chain2    with chain length of two
  chain5    with chain length of five
  chain10   with chain length of ten
  chain20   with chain length of twenty
DETERMINE IF GUARDS ON ENTRY STATEMENTS IMPACT PERFORMANCE
  guard2      with one guard set to true and one set to false
  guard20     with one guard set to true and nineteen set to false
  guard20e    with one guard set to true and nineteen set to false
  guard20et   with all guards set to true
  guards20t   with all guards set to true
  guards2e    with one guard set to true and one set to false
DETERMINE WHETHER IDLE TASKS IMPACT PERFORMANCE
  idle1     with one idle task
  idle5     with five idle tasks
  idle10    with 10 idle tasks
  idle20    with twenty idle tasks
DETERMINE IF IT IS BETTER TO HAVE LOTS OF LITTLE TASKS WITH SINGLE
ENTRY CHOICES OR A FEW BIG TASKS WITH MANY SELECT CHOICES
  moretasks    master task calls twenty slave tasks with single entries
  moretasks1   each task has a single entry embedded in a select
               statement
  moreselct    master task calls each entry in a single slave task
  moreselctr   entries are listed in opposite order from calling order
DETERMINE IF ORDERING OF ENTRY CLAUSES IN A SELECT MATTERS
  order31    with thirty-one choices
  order31r   entries are called in reverse order of select statement
  order32    with thirty-two choices
  order100   with one hundred choices
DETERMINE IF SIZE OF PASSED PARAMETER MAKES A DIFFERENCE
  passarrys   with small "in" array of integers
  passarryb   with larger "in" array of integers
  passinout   with large "in out" array of integers
DETERMINE IF THE NUMBER OF SELECT CHOICES MAKES A DIFFERENCE
  select2     with two choices, desired entry is first
  select2e    with two choices, desired entry is last
  select20    with twenty choices, desired choice is first
  select20e   with twenty choices, desired choice is last
DETERMINE IF ADA SCHEDULER MAY STARVE A TASK
  schedtest   with two entry select statements used independently by
              three other tasks
```

AIE FRONT END TESTS

SYNTAX ERRORS
  (AIE)fen0301a checks frequent comment syntax mistakes
  (AIE)fen0302a checks miscellaneous common spelling mistakes
  (AIE)fen0303a checks common Ada/Pascal errors
  (AIE)fen0304a checks common errors in subprogram specifications
  (AIE)fen0305a checks common errors in subprogram declarations
COMPILER OPTIONS
  (AIE)fen0501a verifies that default options are as specified
  (AIE)fen0502a verifies that specified options take on the given values
  (AIE)fen0503a checks the NOSEM option
  (AIE)fen0503b checks the NOSEM option
  (AIE)fen0503c checks the NOSEM option
  (AIE)fen0503d checks the NOSEM option
  (AIE)fen0503e checks the NOSEM option
  (AIE)fen0503f checks the NOSEM option
  (AIE)fen0504a checks the NOCODE option
  (AIE)fen0504b checks the NOCODE option
  (AIE)fen0504c checks the NOCODE option
  (AIE)fen0504d checks the NOCODE option
  (AIE)fen0504e checks the NOCODE option
  (AIE)fen0504f checks the NOCODE option
  (AIE)fen0505a checks the COMMENT option
  (AIE)fen0505b checks the COMMENT option
  (AIE)fen0506a checks the LOOKAHEAD option
  (AIE)fen0506b checks the LOOKAHEAD option
  (AIE)fen0506c checks the LOOKAHEAD option
  (AIE)fen0506d checks the LOOKAHEAD option
CHECK CORRECT MEANING OF DOT "." AND TIC "'"
  (AIE)fen0701a verify dot as a delimiter in numeric range
  (AIE)fen0701b verify dot as part of a numeric literal
  (AIE)fen0701c verify tic in attribute selection
  (AIE)fen0701d verify tic in type qualification
  (AIE)fen0701e verify tic in character literal
CHECK HIDING
  (AIE)fen0702a by nested blocks
  (AIE)fen0702b by loop parameters
  (AIE)fen0702c by nested packages
  (AIE)fen0702d using the extended scope of a pkg spec through its corresponding body
  (AIE)fen0702e with nested (non-overloaded subprograms
  (AIE)fen0702f within a record type declaration
  (AIE)fen0702g by an object renaming declaration
  (AIE)fen0702h by nesting a subprogram declaration
  (AIE)fen0702i by a formal parameter of nested subprogram declaration
  (AIE)fen0702j by nesting a task definition
  (AIE)fen0702k of an object by a subprogram declaration with the same identifier
  (AIE)fen0702l checks hiding of a predefined type from STANDARD
CHECKS OVERLOADING ENTITIES
  (AIE)fen0703a using a procedure and a function
  (AIE)fen0703b using procedures with different numbers of parameters
  (AIE)fen0703c using functions with different numbers of parameters

```
  (AIE)fen0703d using procedures with different base types of
                parameters
  (AIE)fen0703e using functions with different base types of
                parameters
  (AIE)fen0703f using functions with different return base types
  (AIE)fen0703g using procedures with differently ordered parameters
  (AIE)fen0703h overloading an operator
CHECKS USE CLAUSES
  (AIE)fen0704a using two nested packages, one which is USEd
  (AIE)fen0704b using nested package which has an identifier that
                should not hide a directly visible identifier
  (AIE)fen0704c using nested packages with dot selection
  (AIE)fen0704d using separately compiled pkgs using WITH and USE
CHECKS VERY LARGE SOURCE LINES
  (AIE)fen0801a verifies line length of 255 allowed
  (AIE)fen0801b verifies line length of >255 flagged
CHECKS LARGE NUMBERS OF VMM SUBDOMAINS
  (AIE)fen0803a opens 198 subdomains using a WITH chain 197 units
  (AIE)fen0803b opens 199 subdomains using a WITH chain 198 units
  (AIE)fen0803c opens 200 subdomains using a WITH chain 199 units
  (AIE)fen0803d opens 199 subdomains using a WITH chain 196 units
                with 2 nested units
  (AIE)fen0803e opens 200 subdomains using a WITH chain 197 units
                with an indirect WITH and a subunit
CHECK OVERFLOW OF PARSE STACK OR STATE STACK OF THE PARSER
  (AIE)fen0804a using nested block statements level 5
  (AIE)fen0804b using nested block statements level 10
  (AIE)fen0804c using nested block statements level 15
  (AIE)fen0805a using nested loop statements level 5
  (AIE)fen0805b using nested loop statements level 10
  (AIE)fen0805c using nested loop statements level 15
  (AIE)fen0806a using nested if statements level 5
  (AIE)fen0806b using nested if statements level 10
  (AIE)fen0806c using nested if statements level 15
  (AIE)fen0807a using nested package statements level 5
  (AIE)fen0807b using nested package statements level 10
  (AIE)fen0807c using nested package statements level 15
  (AIE)fen0808a using nested subprogram statements level 5
  (AIE)fen0808b using nested subprogram statements level 10
  (AIE)fen0808c using nested subprogram statements level 15
  (AIE)fen0809a using WITH, USE, variable decls, arithmetic and
                logical expressions which are not quickly resolvable
```

CHECKS GENERIC INSTANTIATIONS
(AIE)mid0301a verify Diana is not modified if no generic instantiation
(AIE)mid0302a verify instance body created for instantiation is correct Diana

CHECK GENERICS COMPILED AT DIFFERENT TIMES IN VARIOUS ORDERS
(AIE)mid0303a generic procedure declaration, body and instantiation (single compilation)
(AIE)mid0303b generic procedure declaration, instantiation, and body (single compilation)
(AIE)mid0303c generic procedure declaration, body and instantiation (separate compilation)
(AIE)mid0303d generic procedure declaration, instantiation and body (separate compilation)
(AIE)mid0303e generic procedure declaration, its package body containing the body and an instantiation - main proc using the instantiation (separate compilation)
(AIE)mid0303f generic procedure declaration, its package body containing the generic body stub and an instantiation, the generic body subunit (main proc using the instantiation)(separate compilation)
(AIE)mid0304a generic package declaration, body and instantiation (single compilation)
(AIE)mid0304b generic package declaration, instantiation, and body (single compilation)
(AIE)mid0304c generic package declaration, body and instantiation (separate compilation)
(AIE)mid0304d generic package declaration, instantiation and body (separate compilation)
(AIE)mid0304e generic package declaration, its package body containing the body and an instantiation - main proc using the instantiation (separate compilation)
(AIE)mid0304f generic procedure declaration, its package body containing the generic body stub and an instantiation, the generic body subunit (main proc using the instantiation)(separate compilation)
(AIE)mid0305a generic procedure declaration, instantiation, and main procedure (single compilation)
(AIE)mid0305b generic package declaration, instantiation and main procedure (separate compilation)
(AIE)mid0306a generic package declaration, instantiation, and main procedure (single compilation)
(AIE)mid0306b generic package declaration, instantiation and main procedure (separate compilation)

CHECK SHARING OF GENERIC BODY INSTANTIATION REPRESENTATIONS
(AIE)mid0307a check that 'in' actual parameters match
(AIE)mid0308a check that 'in out' actual parameters match
(AIE)mid0309a check that integer type actual parameters match
(AIE)mid0310a check that floating point type actual parameters match (verifies when they have the same amount of storage)
(AIE)mid0310b check that floating point type actual parameters match (verifies if they have identical rep specs)
(AIE)mid0311a check that fixed point type actual parameters match

```
                  (verifies when they have the same amount of storage)
   (AIE)mid0311b check that fixed point type actual parameters match
                  (verifies if they have identical rep specs)
   (AIE)mid0312a check that discrete type actual parameters match
                  (verifies when they have the same amount of storage)
   (AIE)mid0312b check that discrete type actual parameters match
                  (verifies if they have identical rep specs)
   (AIE)mid0313a check that access type actual parameters match
 CHECK CROSS REFERENCE INFORMATION TO si_refs ATTRIBUTE OF DEF_IDs
   (AIE)mid0401a which are subtypes, types, and variables
   (AIE)mid0401b verify no information is added when LIST => NOXREF
   (AIE)mid0401c which are procedures, functions, operators,
   (AIE)mid0401d which are labels and packages
   (AIE)mid0401e verifies references are not included from uses in
                  other units
   (AIE)mid0401f when LIST => XREF
 CHECK CROSS REFERENCE INFORMATION TO si_calls ATTRIBUTE OF DEF_IDs
   (AIE)mid0402a when subprograms invoked from within subprograms
   (AIE)mid0402b check no information is added when LIST => NOXREF
   (AIE)mid0402c when subprograms and operations invoked within
                  subprograms, operations and packages
   (AIE)mid0402d when invocations from a package specification and
                  body
 CHECK ACROSS REFERENCE INFORMATION TO si_external_refs ATTRIBUTE OF
   A COMPILATION UNIT NODE
   (AIE)mid0403a using a procedure compilation unit
   (AIE)mid0403b verify not si_external refs added if LIST => NOXREF
   (AIE)mid0403c using package compilation units
   (AIE)mid0403d using subunit compilation unit
   (AIE)mid0403e using a function body compilation unit
   (AIE)mid0403f using a procedure specification compilation unit
 CHECK CROSS REFERENCE INFORMATION TO si_global_refs ATTRIBUTE OF A
   BLOCK OR BODY NODE
   (AIE)mid0404a using a procedure body
   (AIE)mid0404b verify no information idded if LIST => NOXREF
   (AIE)mid0404c using a function body
   (AIE)mid0404d using a block statement
   (AIE)mid0404e using a package specification and body
 VERIFY VALUE GIVEN TO THE si_labeled ATTRIBUTE OF STATEMENTS
   (AIE)mid0405a using assignment, goto and null statements
   (AIE)mid0405b using if, loop and exit statements
   (AIE)mid0405c using block and case statements
   (AIE)mid0405d using procedure calls and return statements
 VERIFY VALUE GIVEN TO THE si_context ATTRIBUTE OF NAME_EXP NODES
   (AIE)mid0406a those that should receive ADDRESS_CONTEXT value
   (AIE)mid0406b those that should receive FLOW_CONTEXT value
   (AIE)mid0406c those that should receive PARAMETER_CONTEXT value
   (AIE)mid0406d those that should receive VALUE_CONTEXT value
 VERIFY VALUE GIVEN TO THE si_opt_level ATTRIBUTE OF A BODY OR BLOCK
   (AIE)mid0501a using procedure body without a pragma
                  (OPTIMIZE=>NONE)
   (AIE)mid0501b using procedure body without a pragma
                  (OPTIMIZE=>TIME)
   (AIE)mid0501c using procedure body without a pragma
                  (OPTIMIZE=>SPACE)
   (AIE)mid0502a using procedure body with pragma OPTIMIZE(SPACE)
                  (OPTIMIZE=>TIME)
```

(AIE)mid0502b using block statement with nested pragmas OPTIMIZE(SPACE) and OPTIMIZE(TIME) (OPTIMIZE => NONE)
(AIE)mid0502c using package spec and body with pragma OPTIMIZE(TIME) (OPTIMIZE=>SPACE)
(AIE)mid0502d using nested block statements with pragma OPTIMIZE(SPACE) (OPTIMIZE=>NONE)

CHECK IMPROPER USES OF PRAGMA OPTIMIZE
(AIE)mid0503a using second pragma OPTIMIZE in a declarative part (OPTIMIZE=>NONE)
(AIE)mid0503b pragma OPTIMIZE cannot be placed in package specification

VERIFY THE AIE DEFINED PRAGMA STATIC
(AIE)mid0504a verify call frame for each of subprogram arguments is allocated static storage
(AIE)mid0504b verify error when there is dynamically sized local

VERIFY PRAGMA INLINE
(AIE)mid0505a verifies subprogram bodies are expanded inline

VERIFY DECISIONS MADE WHEN CHOOSING A LAYOUT REPRESENTATION FOR STORAGE
(AIE)mid0601a predefined type INTEGER given a layout of one word
(AIE)mid0601b predefined type SMALL_INTEGER given a layout of 1/2 word
(AIE)mid0601c verify user defined integer types given smallest layout
(AIE)mid0602a predefined type FLOAT is represented as single word
(AIE)mid0602b predefined type LONG_FLOAT is represented as single word
(AIE)mid0602c user defined floating point types are given smallest layout
(AIE)mid0602d error issued when accuracy requested is too precise for implementation
(AIE)mid0603a fixed point types represented as single word
(AIE)mid0603b or issued when accuracy requested is too precise for implementation
(AIE)mid0604a object represented with 3 bits is given 3 bits in packed record
(AIE)mid0604b object represented with 3 bits is given a byte in an unpacked array
(AIE)mid0604c object represented with 3 bits is given 1/2 word as a local ojbect to a subprogram
(AIE)mid0605a statically sized components are stored in the beginning of record and dynamically sized components at end with pointer from beginning
(AIE)mid0606a representations are as small as possible in packed array
(AIE)mid0606b representations are expanded to byte to ease addressing in unpacked array
(AIE)mid0607a storage for variant parts of records is overlaid
(AIE)mid0607b storage for disjoint blocks within subpgm bodies is overlayed
(AIE)mid0608a verify allocation on secondary stack for dynamic sized arrays not in records
(AIE)mid0609a checks length clauses for numeric type (rep specs)
(AIE)mid0609b checks length clauses for access type (rep specs)
(AIE)mid0609c checks length cluases for task type (rep specs)
(AIE)mid0610a checks record rep clause that specifies the packing

in 2 words

VERIFY STATICALLY SIZED OBJECTS SIZED <= 64 BITS ARE USED AS FORMAL PARAMS - ACTUALS PASSED BY VALUE

(AIE)mid0701a checks parameters of procedures

(AIE)mid0701b checks parameters and return values of functions

VERIFY STATICALLY SIZED OBJECTS SIZED >= 64 BITS ARE USED AS FORMAL PARAMS - ACTUALS PASSED BY VALUE

(AIE)mid0702a checks parameters of procedures

(AIE)mid0702b checks parameters and return values of functions

VERIFY CORRECT PARAMETER PASSING METHODS ARE SELECTED

(AIE)mid0703a checks parameters of procedures

(AIE)mid0703b checks parameters and return values of functions

VERIFY DYNAMICALLY SIZED OBJECTS USED AS FORMAL PARAMS PASSED BY REFERENCE

(AIE)mid0704a checks parameters of procedures

(AIE)mid0704b checks parameters and return values of functions

AIE BACK END TESTS

CHECK COMPILER OPTIONS

ben0101a verify memory size with OPTIMIZE => SPACE
ben0102a verify cpu time with OPTIMIZE => TIME
ben0201a check FLOW with OPTIMIZE => NONE
ben0202a check strength reduction and code motion when OPTIMIZE => SPACE

CHECK FLOW AND WALKS OF BILL TREE

ben0203a with LOOP and no subprogram calls
beno2o3b with GOTO and no loops or subprogram calls
beno203c strength reduction unit processing
ben0203d with subprogram call and INLINE pragma
ben0203e with subprogram call, subprogram defined in program
ben0203f with subprogram call, subprogram declared in package
ben0203g with subprogram call, subprogram in library unit
ben0203h with subprogram call, recursive subprogram in program
ben0203i with subprogram call, recursive subprogram in package
ben0203j with subprogram call, recursive subprogram in library
ben0203k with GOTO, no loops or subprograms, detection of common subexpressions
ben0204a with units eligible for strength reduction when OPTIMIZE => TIME
ben0205a with straightline program

CHECK CONSTANT PRAPAGATION WITH EXPRESSION SIMPLIFICATION

ben0301a when constant is a named number
ben0301b when constant is a true constant

CHECK FOLDING IMPLEMENTATION BY FLOW

ben0302a where expression has binary operation and integer operands
ben0302b where expression has unary operation and integer operands
ben0302c where expression has deeply nested unary operations
ben0302d where boolean expression has unary or binary operations and boolean operands
ben0302e where constants are adjacent and where they are dispersed

CHECK TREATMENT OF EXPRESSIONS AS COMMON SUBEXPRESSIONS

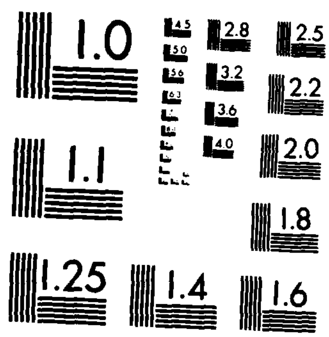
ben0303a with binary expressions and no change of any variable
ben0303b with binary expressions and a change of some variable
ben0303c with components of an array
ben0303d with binary expressions, variable changed in IF and CASE
ben0303e with subexpresions permuted or muddled by parentheses
ben0303f with array offset
ben0303g with array offset and value assigned object
ben0303h with arguments to subtype conversion functions
ben0303i with arguements to type conversion functions
ben0303j with labelled statement to GOTO
ben0303k with labelled statement with no GOTO
ben0303l with nested levels of declaration of subexpressions
ben0303m with record components

CHECK EXPRESSION SIMPLIFICATION

ben0304a with integer , no side effect as a sub_operand
ben0304b with integer, with side effect as an operand

```
ben0304c  with floating point, with side effect as an sub_operand
ben0304d  with floating point,  no side effect as a sub_operand
ben0304e  with boolean, no side effect as a sub_operand
ben0304f  with boolean, with side effect as a sub_operand
ben0304g  with cancelling of subexpressions

CHECK STRENGTH REDUCTION
ben0401a  with FOR loop
ben0401b  with FOR loop, elements not in usual arrangement
ben0402a  with invariant code movement

CHECK RANGE INFORMATION AND UNNECESSARY CONSTRAINT CHECKS
ben0501a  at END IF statement
ben0501b  at END CASE statement
ben0501c  with BACK END check range information
ben0501d  with BACK END check access checks

CHECK REMOVAL OF UNREACHABLE CODE
ben0502a  by FINAL with 'if-else-then' with constant propagation
ben0502b  by FINAL with 'if-else-then' with range information
ben0502c  by FLOW with LOOPS, condition always false or range null
ben0502d  by FINAL with CASE, with constant propagation
ben0502e  with GOTO
ben0502f  between RAISE, with unhandled exception
ben0502g  between RAISE, with handled exception
ben0502h  by FLOW in CASE, where alternative chosen by null range
ben0502i  by FLOW in CASE, don't need OTHERS choice
ben0503a  nested GOTOs changed to one GOTO

CHECK "CROSS-JUMPING" WITH COMPILER OPTION, OPTIMIZE => SPACE
ben0504a  at end of THEN, ELSE
ben0504b  at end of THEN, ELSIF, ELSE
ben0504c  at end of CASE
ben0504d  at end of CASE where several sets of duplicate code
ben0504e  at end of THEN, ELSE where code semantically the same

CHECK PRAGMA INLINE SUBPROGRAM
ben0505a  instead of compilation unit
ben0505b  in declaration of space optimized subprograms

CHECK BACK END USE OF LOCAL SPILL
ben0601a  with integer expression that requires 16-plus registers
ben0601b  with floating . expression that requires 4-plus registers
ben0601c  with complex expression that requires 16-plus registers
ben0602a  with simple expression

CHECK COMPILER BACK END LIMITATIONS
ben0701a  verify speed of compilation
ben0702a  verify size of compiler
ben0703a  verify maximum size of BILL node
```

AD-A153 609 EVALUATION AND VALIDATION (E&V) TEAM PUBLIC REPORT 5/6
VOLUME 1(U) AIR FORCE WRIGHT AERONAUTICAL LABS
WRIGHT-PATTERSON AFB OH V L CASTOR 30 NOV 84
UNCLASSIFIED AFWAL-TR-85-1016-VOL-1 F/G 14/2 NL



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

APPENDIX K

E&V WORKSHOP POSITION PAPERS

Table of Contents

SIMULATION: AN IMPORTANT ISSUE FOR
APSE EVALUATION AND VALIDATION

BARD S. CRAWFORD
THE ANALYTIC SCIENCES CORPORATION

SIMULATION: AN IMPORTANT ISSUE FOR APSE EVALUATION AND VALIDATION

by Bard S. Crawford

1. INTRODUCTION

The thesis of this position paper is that simulation software deserves strong consideration in the design of APSEs and, therefore, in the evaluation and validation of APSEs Simulation software can be viewed in two ways, as follows:

- It can be viewed as part of an APSE, useful in supporting the development and testing of embedded system software.
- It can be viewed as a category of application software, which requires general-purpose simulation support packages that are part of an APSE.

Both of these views are valid and both lend support to the thesis expressed above.

2. THE IMPORTANCE OF SIMULATION

Throughout the entire life cycle of the development and use of embedded software systems, simulation has traditionally played a number of key roles. Furthermore, these roles are likely to increase in importance in the future, especially in a "software

first" scenario or in a weapon system development program in which software "defines the system." The traditional uses of simulation software include:

- Support of early analysis and prototyping of critical elements of an advanced system concept.
- Provision of test driver programs in support of both unit tests and integrated system tests of embedded systems under development.
- Support of software "maintenance" activities including assessment of proposed software changes and acceptance testing of newly-inserted changes.
- Provision of "hardware in the loop" simulators and training simulators.

Across the life cycle of an embedded software system it is not uncommon for the amount of simulation software code to exceed the size of the actual embedded system code. Thus, a good simulation support environment can pay significant productivity dividends in two regimes: during the development of the simulation code itself and during the analysis, testing and maintenance of the embedded code -- by virtue of the greater effectiveness of the simulation packages employed.

3. SIMULATION TOOLS AS APSE COMPONENTS

Figures K-1 and K-2 illustrate two views of simulation software in relation to APSEs. Figure K-1 labeled "Prototyping Scenario" represents a "pure simulation" or scientific programming environment used to perform exploratory investigations of application domains and/or prototype algorithms. The APSE segment labeled "Support Packages" represents, for example, general purpose simulation support tools (Ada packages) such as discrete-event process schedulers, continuous simulation integration

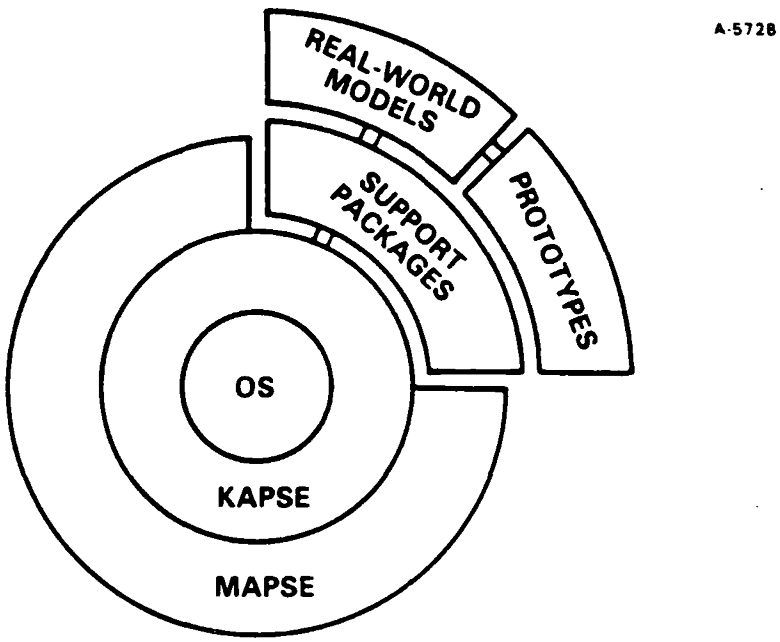


Figure K-1. Prototyping Scenario

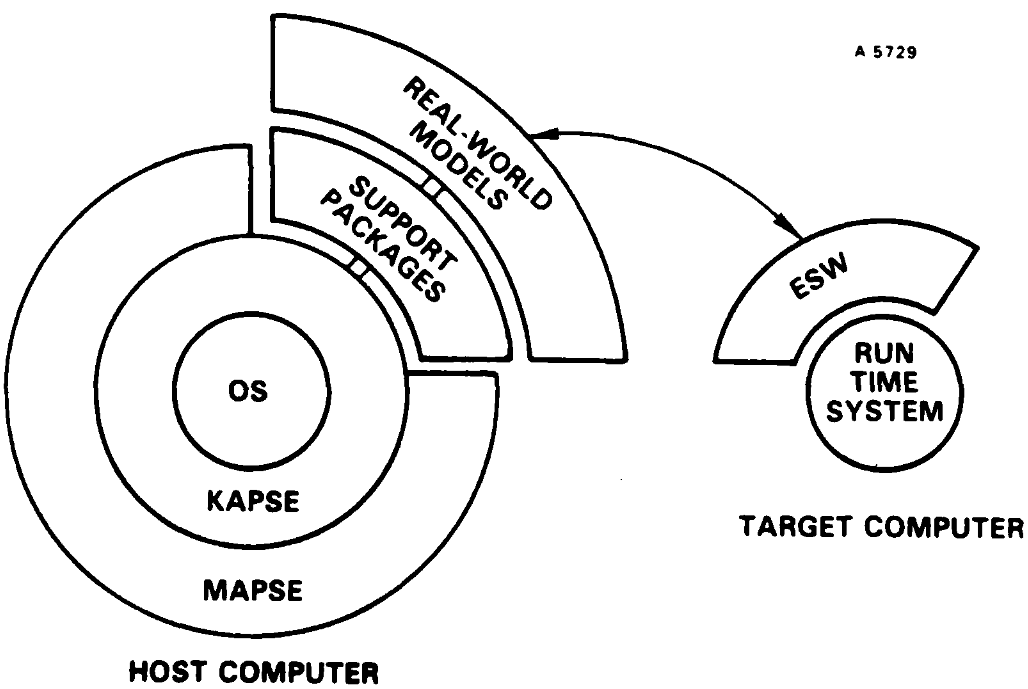


Figure K-2. Testing and Maintenance Scenario

routines or combined discrete/continuous support tools. Two application software segments are shown: "Real-World Models" and "Prototypes." The latter are early experimental versions of modules that will eventually evolve into embedded software in a different scenario.

The former, "Real-World Models," could be viewed in one of two ways, as mentioned in Section 1. In this figure it is pictured as an example of application software -- outside the APSE. It might also be pictured as a part of the APSE -- that is, as a set of tools used to support a specific application area, but serving a spectrum of projects within that area. The choice of viewpoint is somewhat arbitrary. The main point to be made here is that these Real-World Models are an extremely important category of software to be considered in the evaluation of APSEs-- whether they are considered part of the APSE or as the "first level" of application software supported by the APSE.

Figure K-2, labeled "Testing and Maintenance Scenario" represents the "mainline" developmental and operational environments. In this case a host computer supports the APSE and simulation software, which communicates with the developmental Embedded Software (ESW) running on a target computer (or emulator). Again, the Real-World Models are an extremely important element, with significant implications for APSE design, evaluation and validation.

4. EVALUATION AND VALIDATION ISSUES RELATED TO SIMULATION

Listed below are some suggested activities designed to bring out issues related to simulation support tools.

- Identification of existing simulation support tools in existing environments or under study in research programs.

- Identification of missing-but-required simulation-support tools.
- Categorization of simulation-support tools and tool features in a manner useful to E&V.
- Identification of tool interface issues (e.g. CAIS compatibility) relevant or unique to simulation-support tools.
- Development of evaluation criteria (formal and informal) appropriate to simulation-support tools.
- Definition of baseline or benchmark simulation scenarios useful in the evaluation of APSEs.
- Identification of simulation-support tool usefulness as a function of life-cycle phases and methodologies.

Given the crucial role of simulation software and its intimate relation to the APSE outlined earlier, it follows that the issues/activities called out above are of major importance to the APSE E&V Effort.

VALIDATION AND STANDARDS
IN ADA ENVIRONMENTS

PAUL DOBBS
GENERAL DYNAMICS

4.1 Validation And Standards In Ada Environments

Paul Dobbs
General Dynamics
Data Systems Division
Central Center
Fort Worth, Texas

This paper examines those factors within an Ada Programming Support Environment (APSE) which can be validated against a standard, attempts to recommend new standards for development, and examines methods for validating against those standards. It is based on earlier research performed for the Ada Evaluation and Validation (E&V) Committee under the auspices of the Integrated Support Software System (ISSS) contract. Documents which were produced during that research and which will be referred to in this paper include "Evaluation and Validation of a Compiler", "Evaluation Questions", "E & V Tool Features Evaluation", and an untitled listing of tools and interfaces.

Compiler Compliance with the Ada Standard

It is important that all Ada compilers be validated with respect to the Ada language standard. This is already being done with the use of the Ada Compiler Validation Capability. However, there are still some unresolved issues dealing with validation. One of these deals with the standard I/O packages. Many embedded computer systems do not do any form of text I/O, nor do they need a file system. An example of such a system is a flight control system. Its inputs are switch positions, control pressures, acceleration reading, indications of whether or not the wheels in the landing gear are spinning, and so on. Its outputs are signals to the servos controlling the positions of the flight control surfaces. It would seem reasonable that a company which was having a compiler written for this application should not have to pay for having packages written for text, random, and sequential I/O, when the only use for these packages is to get

the compiler through validation. It is even a possibility that the hardware of the target computer might not support these types of I/O. A study needs to be conducted to determine whether it would be possible to relax the language requirements for the standard packages. One possible problem is that removing these packages leaves no simple way to obtain results from those tests in the validation suite which actually execute. It might be possible to require that the organization requesting the validation of a compiler which does not provide the standard I/O packages provide the means by which the results of executing tests can be determined.

Tool Compliance with CAIS

The Common APSE Interface Set (CAIS) is a standard set of KAPSE interfaces which is under development by the KAPSE Interface Team (KIT). Compliance with this standard will become mandatory when it is fully developed. The standard is in the form of Ada package specifications and descriptions of semantic actions. A suite of test programs could be developed which would exercise the interfaces of a APSE and check it for compliance. In this way, each APSE could be validated. A technical report on "Validation in Ada Programming Support Environments" has been written by a team from Virginia Tech and MITRE, so CAIS standardization will not be further discussed in this paper.

Compiler and Tool Compliance with Diana

The Distributed Intermediate Attributed Notation for Ada (Diana) is a proposed intermediate language for Ada compilers. It is not yet a standard, but it has been suggested that it be eventually made into a standard. The advantage of having a standard intermediate language is that, if it were rigidly enough defined, compiler front ends could output Diana which could be stored for use by other tools, as well as by the compiler back end. Tools which might be able to use Diana include PDL processors, code formatters (pretty printers), optimizers, statistical profilers, and of course the compiler itself. A standardized intermediate language makes it possible to construct these tools without reference to any specific environment, so that they can be transported from environment to environment

easily. In addition, it would be possible for third parties to develop code optimizers and new back ends (code generators) in a reasonably portable fashion if a common intermediate language existed.

The best form for such a common intermediate language standard would be Ada package specifications and descriptions of standard semantics. The package specifications would have to cover both the type definitions (implementation details could be private) and accessing procedures (whose implementation dependent details could be hidden in package bodies). Each environment could provide a set of implementations (package bodies and the private parts of the specifications) for the Diana package specifications, which would solve the problem of making the intermediate language efficient on different machines. A tool which was being imported into a new environment would have to be recompiled using the implementations on the new environment, but it would need to be recompiled anyway to conform to the local implementation of the CAIS.

A pair of test program suites would need to be developed to test compliance with the intermediate language standard. One of the suites would test compliance by programs that produce Diana. It could consist of Ada programs which would be fed to Diana generators to produce Diana which could be compared with the expected results. The second suite would provide Diana which would be fed to those programs which use Diana as input. The problem with the second suite of tests would be that there are many types of Diana-using tools, and even for tools of the same type, their output is usually not subject to standards. That means that checking for compliance would be a costly manual process. Because of the time and effort involved in checking the output, and the fact that this process could not be effectively automated, it is not reasonable to validate the compilance of Diana-using tools to the standard.

Text Files

As strange as it may sound, it may be necessary to issue a standard on text files. Although "everyone" knows what a text file is, actual examination of the situation reveals that not all text files are created equal. Some text

editors store compressed text files in which blanks are replaced by blank counts. An example of such an editor is the Wilbur system on the IBM 370 (Not an IBM product), which replaces every occurence of one or more blanks with a byte in which the first four bits represent the number of blanks, and the next four bits represent the number of nonblank characters before the next blank. These files must be expanded by Wilbur before being input to a compiler. This is an extreme example, but even among "pure" text files, there are differences. Most of these differences involve the use of control characters. Many systems use tab characters to save space in text files. Unfortunately, they do not all agree on just what the tab represents. Many terminals are set up with predefined tabs every eight spaces, and will accept tabs in text to be displayed, but some react to the tab as though it were the end of the line. End-of-lines are another problem. On some systems, end-of-line is a carriage return, on some it is a line feed, and on others it is a combination of both.

These are small details, but such problems as these can be extremely frustrating to the user of a system where they have not been taken into account. Therefore, it would be good if such details were thought out in advance and set into writing as a standard. Validating such a standard should be fairly simple, since it should only involve checking text files output by the system tools. If the standard is properly constructed, it should be possible to fully automate this process.

Compiler Output

There is a minor need for standardization of compiler outputs. These may consist of object files or assembly language files. The problem is that multiple assembly languages exist for the same processors, and that the same is true for object files. It does little benefit to have a MIL-STD-1750A Instruction Set Architecture, for example, and have tools sets (TRW and MDAC) which have different object file formats. A standard should be chosen. However, there is no real need for validation of these standards.

Conclusions

From the above discussion, there appears to be some need for standards in the areas of intermediate language (Diana), text files, and compiler output files. This is an area in which the E&V committee should concern itself. Also, the validation standard for the Ada language itself may need to be reexamined in terms of the real needs of the embedded computer community. However, this is more an AJPO matter than an E&V committee matter, as things currently stand.

ENVIRONMENT EVALUATION AND VALIDATION:
THE USER'S PERSPECTIVE

ROBERT E. FRITZ
COMPUTER SCIENCES CORPORATION

ENVIRONMENT EVALUATION AND VALIDATION: THE USER'S PERSPECTIVE

Robert E. Fritz
Computer Sciences Corporation
Applied Technology Division
4045 Hancock Street
San Diego, CA 92110

Introduction For software developers the criteria for evaluating tools and environments are simple: the environment is there and helps them do what needs to be done. Anything less that this is inadequate, for the purpose of the environment is to aid the software engineer produce programs not prove the validity of an idea. The Ada Programming Support Environments are in particularly sensitve position because of the importance of the systems they will be used to produce, and because of the high visibility of the Ada program. Strong standards meaningful to software engineers to ensure that APSEs will be available and usable must be created. The qualities of a usable environment are easily described, but quantifying these criteria is a major issue. The technology applied to achieve the usability criteria may be greatly different.

Availability. No tool or environment is useful if it does not exist or is not available on the system used for project devlopment. Many environment tools are meaningless if certain core tools are not available. For the APSE, the essential tool is the Ada compiler. If the compiler is not adequate for the task no amount of tools or sophistication will make the project succeed if implementation cannot be done. After the compiler the second most important tool is the file management or support for separate compilation. Without these tools in place there can be no progress.

Other tools such as configuration managers, editors, design aids and others provide convenience but are useless without the compiler. Environment construction should be done incrementally after an overall design has been done to a moderate level of detail for the range of life-cycle support tools desired. This implies a prioritization of tools with a core tool or tool described for the various life cycle stages, as well as an assignment of priorities to various life-cycle stages and activities. The Software Engineering Automation for Tactical Embedded Computer Systems (SEATECS) project of the U.S. Navy has attempted such a prioritization [1] which could serve as a point of departure for a generalized prioritization for APSE tools. The priority levels used were near-term requirements, mid-term requirements, and long term requirements according to the criticality for support of Navy systems and technological feasibility. Criticality had three levels of classification:

a. Immediate. Essential support of basic mission.

b. Productive. Capabilities would significantly improve productivity. These capabilities would are initially supplemental but become essential as workload grows.

c. Desirable. These capabilities enhance the ability to perform the mission but the absence of which will not be detrimental to performance of mission.

Technological feasibility describes the ease with which a tool may be built or incorporated into an environment. The feasibility classifications used by SEATECS are:

a. Available. Capability available off-the-shelf and requires little adjustment.

b. Convertible. Capability technologically available, but not in the exact form required. Some modification is necessary.

c. Experimental. Capability has been demonstrated experimentally or in limited form. Risk is involved to complete theory or implementation of capability for use in the environment.

d. Theoretical. Theory or preliminary groundwork exists but no automation.

e. Unknown. No theory, research, or automation is known for the capability.

For assigning priorities to desired environment capabilities, other criteria may also be developed which better reflect the needs of the users of the environment. Quantification of the criteria is also desirable where possible to provide better boundaries for environment production.

Once priorities have been assigned, production schedules for environments should be assigned to build the tools according to priority where possible. However some tools will have to be built of priority order because of technical requirements of other tools. The production of a lower priority antecedent tool should not disrupt production of unrelated higher priority tools. Schedule engineering for APSEs must be done to insure that the current trend of APSE development is not maintained: production of large monolithic environments at the expense of core tools, specifically the Ada compiler and related tools.

<u>Usability</u>. The criteria for user satisfaction with a particular tool or a general environment may be enumerated conceptually but it is more difficult to provide quantitative measures. The usability of a tool is measured subjectively by different users with different levels of skill, experience, training, intent, and with diferent personalities. The criteria for user approval [2] include:

- <u>Functionality</u> The range of tasks that the user can do with the system.
- <u>Learning</u> How long it takes to learn how to do a given set of tasks.
- <u>Speed</u> How long it takes the user to do a given set of tasks.
- <u>Errors</u> How many errors the user makes and how severe they are.

- Quality — How good the output of a given task is.
- Robustness — How well the user and tool adapt to new and unexpected tasks.
- Acceptability — How well the user subjectively rates the system for doing a given set of tasks.

Other criteria may be added to this list:

- Integration — How well individual tools work together.
- Human Interface — How physically and psychologically ergonomic the tools or system are.

Functionality. A tool should be limited to solving as small a piece of the problem as is reasonable. The function that the tool automates should be based on the user's structuring of the problem, and not on the software system architecture.

The environment must be able to aid the user in solving the range of problems for which the environment was designed. Adding functions which are "nice" because they allow the environment to be extended to other areas should not be done if the addition detracts from the essential purpose of the environment.

Learning. Tools should be easy to learn to use and protect the user from harming himself during this period. On-line tutorials which lead the user through basic command sequences and on-line help facilities help the user get started quickly and give him confidence in the productivity of the tool. The most heavily used features of the tool should be the easiest to use. The tool should be constructed to reflect the user's way of doing things. The tool is meant to aid the user, the user is not meant to aid the tool.

The greatest aids to learning environments are consistency and functional independence of tools. Commands with similar functions, such as editing, should be identical or be similar throughout the environment. Tools should not overlap but have a clearcut purpose and well-known interfaces to other tools. Help facilities which describe the use of tools so the user may choose the best one increase the user's ability to use the environment well.

Both individual tools and environments should support users with different levels of experience. Novice users need prompting and menu driven systems to provide a lot of support. Expert users must have the capability to circumvent time-consuming menus by using some sort of quick mode for the tool or environment commands.

Speed. A user views the speed of a tool in two ways: how long it takes to react to a specific input and how long it takes to perform a general function. Generally, users expect very fast response to opening or closing

commands and for data editing. Users will accept longer times for processes such as compilation and code generation, or data base restructuring.

Errors. A tool should prevent the user from entering erroneous data or illegal commands. Commands which result in the deletion or radical change of data files should be questioned. The user of the tool should be made aware of the consequences of the action and forced to confirm the request. Command names should be consistent with those of other tools, and be intuitive so far as possible to allow easy learning.

When errors do occur, a simple explanation of the error should be produced, not a cryptic error code which must be referenced in a manual. Error messages should be on-line and have options for tutorial explanations of tool commands. Means for error recovery should be provided, including creation of backup files accessible to the tool user.

System command language errors should be handled similarly to tool errors. Help facilities should be available to list commands, with brief explanations of purpose and syntax available at a lower level, and a full tool tutorial at an even lower level. User roles can be used to control access of users to tools and data to prevent undesirable access or modification.

Quality. The tool must accomplish the desired function and produce output which is useful to the user. The output may be in a form to be passed to another tool, or it may be in a human-readable form.

Robustness. Tools should be adaptable to many analogous activities. For example, the editor used for word processing shouyld have a mode for program editing. A scheduling tool should work as well for a fixed-price C3-I project as it does for a cost-plus avionics project, though with different inputs. Tools which operate with templates should be supplied with several likely candidate templates. Template creation should be supported with tools and tutorial interaction.

The APSE must be able to support development and maintenance of embedded computer systems. This was the design objective for Ada. The APSE should also be capable of supporting other uses, including training and education, scientific processing, business data processing, and communication and networking. These are secondary needs and should not detract from the primary purpose.

Acceptability. The user must like the tool or system. The tool and system must gain his confidence by providing the function and efficiency he seeks to help him in his work. The tool and system must give the user help when needed, protect him from himself, and help the user apply the tools to new situations. The user must be comfortable with the way the system works, and the system must be adaptable to the user's way of doing business. The user must feel that the system is helping him do his work, rather than the user is helping the system do its work.

Integration. The tools within the environment must work well together, in both expected and unexpected combinations of tools. Data formats must be compatible. Functions contained in tools should not overlap. For example, it should be necessary to sort an already sorted file to extract a data

item.

Human Interface. The means by which the user interacts with a tool or the system is the single most important aspect of user evaluation. Both hardware and software contribute to the human interface, and both should be carefully designed. The interface must be adaptable to a number of different styles, and allow for varied amounts of experience. Help should be available in varying levels of detail from brief reminder to complete tutorial. The tool commands and system commands should be consistent and identical where appropriate.

Hardware interfaces may include more than keyboards and CRTs. For some applications alternate I/O devices including mice, touch-screen, trackball, or foot-pedal for input, and high-resolution graphics, color, sound, plots and graphs, video for output.

Architecture. APSE architectures must support a variety of configuration. At the time fo Stoneman, the implicit understanding of the software development system was a minicomputer with a number of terminals. Since that time local area networks and microcomputer work stations have been widely used. Distribution of capabilities between intelligent workstations and larger computers in a network or distributed system is an important consideration to the portability of the environment.

Conclusion The only valid evaluation of an APSE is the user's perception of it. Unfortunately, user evaluation is too late. Those criteria which affect the user's perception of the environment's value should be the primary basis for evaluation of APSEs. Other aspects such as adherence to CAIS guidelines should have only secondary importance.

---

[1] SEATECS Top Level Requirements, NOSC, August 1982

[2] "An Applied Psychology of the User", Thomas P. Moran, ACM Computing Surveys, Vol. 13, No. 1, March 1981, pp. 1-11

EVOLUTIONARY DEVELOPMENT OF AN APSE E&V CAPABILITY

KATHLEEN GILROY
HARRIS CORPORATION

2.0 POSITION PAPER

"Evolutionary Development of an APSE E&V Capability"

Introduction

The goals of transportability and interoperability of Ada Programming Support Environment (APSE) toolsets and databases are promoted through the use of the Common APSE Interface Set (CAIS) currently under development by the KAPSE Interface Team (KIT), and their counterpart from Industry and Academia (KITIA). The Ada Joint Program Office (AJPO) has expressed the opinion that use of the CAIS will eventually be required, with Version 2.0 of the CAIS to be a MIL-STD maintained by the Department of Defense (DoD). A CAIS Validation Capability is the logical starting point in the development of the more general APSE Evaluation and Validation (E&V) Capability which is necessary to accomplish the overall goals of the Ada program.

Evaluation of an APSE implies some measurement of the performance and/or quality of an APSE. This includes the ability to assess qualities such as usability, reliability, and maintainability. The current state-of-the-art is not sufficiently mature to support the setting of standards in this area at the current time. Validation is the determination of conformance of the implemented system to the stated requirements of the system. Conformance to the CAIS will not, however, ensure that programs are either transportable or interoperable. Implementations must possess other qualities to ensure that these types of requirements are met. Validation is the area with which we have a greater understanding and experience, and from which we can build on our previous knowledge incrementally, in much the same manner as the ACVC has and will evolve.

The recommended approach is the incremental development of a layered set of validation suites, which mirrors the conceptual model of the APSE itself. A CAIS Validation Capability would first be developed, with CAIS in this sense meaning the basic virtual operating system facilities required by all APSE tools. Once this capability has been established, a more general tool validation capability would be developed which provides the means for validating additional CAIS tools which are to be added to the system. This will support both the evolution of the CAIS into a more extensive standard tool set, as well as provide a means for standardizing the addition of non-CAIS tools to the APSE.

One of the requirements of the initial CAIS validation process is the ability to monitor the performance of the system. The data resulting from the validation process will provide the beginnings of an evaluation capability. The information collected (from the validation process, or through the monitoring facility in general) could be analyzed and disseminated. Performance is not synonymous with quality, however, and more work will need to be

done to develop criteria for other evaluation classes.

CAIS Validation Capability

The current version of the CAIS is essentially a KAPSE level interface, in that it provides support for the run time system, I/O facilities, and file management. Validation of the current CAIS facilities will be quite different from validation of the types of tools which will rest on top of these basic support facilities. It is the performance of this basic interface set which is the most important. All other tools which are added to the system must interface with this set.

Determination of conformance of a CAIS implementation requires that a set of interface requirements be established from which validation may be performed. Currently, the stated conformance requirements established by the KIT/KITIA in Version 1.1 of the CAIS are a package-by-package syntactic and semantic conformance, and support for minimum pragmatic limits. It is intended that the CAIS could be introduced in an incremental manner, such that subset CAIS implementations could exist during the transition from existing APSEs (most notably the ALS and the AIE) to the CAIS.

A precise description of the semantics of the interface is critical to the development of a validation test. Selection of the test cases depends heavily on the proper specification of the functionality of the unit, since test cases are derived from the analysis of this specification. The first draft of the CAIS contains an initial definition of the semantics of the interfaces, but more work is required if a useful validation capability is to be developed.

This APSE evaluation and validation effort should attempt to develop, in coordination with the KIT/KITIA, those specific conformance requirements which must be validated. A Requirements and Criteria (R&C) document is currently under development by the KIT/KITIA which addresses APSE requirements for achieving transportability and interoperability and which would be applicable to CAIS Version 2.0. A CAIS Implementor´s Guide is also planned which will provide assistence in the development of CAIS implementations which are consistent with the rules for CAIS conformance. Furthermore, those requirements should be of a nature which simplify the validation process. It is proposed that the development of the CAIS and the development of the validation capability could and should proceed in parallel. This could include the development of a Validator´s Guide.

Parallel development of the CAIS and the APSE E&V capability would employ structured analysis techniques in evolving a structured CAIS specification. Structured specifications are expressed in a form that allows analysis of the properties of the specification. Structured specifications allow for inspection of missing information or poorly specified information using both manual and automated means.

This approach would provide a formal methodology for expression of the CAIS requirements in terms of the conceptual system model, and provide evaluation methods for the requirements stated in that conceptual model. Additionally, this approach provides system quality goals, and support for evolution of the system (changes in requirements occurring in future system upgrades).

The CAIS validation process must include the use of a performance monitoring facility, which could possibly also serve as a general purpose software development tool in addition to use for validation purposes.

Tool Validation Capability

Development of an APSE should be accomplished using a system development approach, requiring the specification of a complete and unambiguous set of requirements. Stoneman defines the overall top-level system requirements for an APSE. A more specific set of requirements is needed for the development of a corresponding validation suite. Operational requirements including user scenarios should be defined, followed by specification of other interfaces such as the host/target interface. Data objects and operations on the data objects should be defined next. Finally, the algorithmic requirements of the APSE should be elaborated. Concurrently with these activities a system test plan and procedures would be defined which would validate the APSE by outlining a progressive system test which would begin with individual tool tests, proceed to functional testing, and conclude with APSE system tests.

Validation should be done in a manner similar to current system testing practices, where the test case generation is driven by user scenarios. A unit test similar to the ACVC should be run first to test the individual tool. Next, a functional test where only the tools and protocols necessary for a particular user scenario are included should be performed. Finally, a system test should be used to validate APSE system requirements such as transportability, interoperability, and reliability. New tools can be added to the system through repetition of the system development process. Another unit test is developed for the tool being added. The functional tests are modified to accomodate the new tool. The system test is similarly modified and repeated.

Initial APSE implementations will be composed of the CAIS plus a small set of tools essential to the software development process, i.e., the MAPSE level interface. It is assumed that the CAIS will evolve to absorb many of these tools as they become standardized, but prior to this, each new tool will be built on top of the CAIS. The tool may interface with the CAIS only, or may interface with other tools at this same level, or may interface with a user of the APSE.

Validation of these tools includes assurance that the tool never [illegible] facilities at a lower level than the CAIS, and that the

use of the facilities is consistent with their definition. Validation of these and other APSE tools will additionally require validation of the protocols used to transfer information among the tools, some of which employ complex syntaxes and semantics. Validation of a tool may be complicated by interaction with a user. The protocol definitions could potentially vary widely from tool to tool and from implementation to implementation. It is in the interest of achieving transportabiliy, interoperability, and a validation capability to define a standard set of protocols which must be used for intertool communication and tool to user communication.

The basic user interface to the APSE is through the command language interpreter. The development of a standard command language syntax and semantics is crucial to the goals of interoperability and transportability.

In addition to a command language interface between tools, the CAIS provides means for intertool communications through a message facility, an interrupt signalling facility, process synchronization facilities, and file management facilities. Semantics for the use of facilities such as the interrupt signalling facility are defined in more specific terms than the possible uses of the contents of a data file. It is suggested that a standard set of object-oriented packages encapsulating typical data structures (e.g., lists, sets) be defined on which all tool interfaces must be based. These packages would replace the file management facility as the interface manager. The validation process is simplified by a more rigorous semantic definition of protocols, which limits the number and type of valid intertool interfaces.

Each tool should have its own validation suite, much like the compiler validation suite. The ACVC is an example of validation of a tool as an isolated entity. A complete validation of the compiler would include validation of the editor to compiler protocol (expected contents of an Ada source file), the compiler to library manager (TBD standard library unit definition), and the compiler interface to the CAIS facilities (e.g., to open the source file), among others.

The addition of non-CAIS tools to the APSE would proceed in the same manner. Some definition of "CAIS-conforming" tools should be developed which indicates that the use of the standard CAIS interfaces are not by-passed by the new tool, or the operation of the CAIS modified by its inclusion.

Conclusion

The development of an APSE Evaluation and Validation Capability should be evolutionary.

Beginning with the initial version of the CAIS, a validation suite should be developed for determination of conformance by implementations of the virtual operating system facilities of the

CAIS. The validation suite should be developed concurrently with a complete specification of the CAIS interface requirements. The CAIS Validation Capability requires performance monitoring, with the by-product information used as the basis for development of the evaluation capability.

Once this has been accomplished, the toolset could be incrementally extended and validated using a system development approach. Validation must include not only the testing of the tool in isolation, but also as part of a set of tools and the intertool interfaces. Using the system approach, the higher level requirements of interoperability and transportability can be validated.

Not enough is currently known about evaluating APSEs for quality requirements such as usability, and these capabilities will have to be developed as more is learned.

ADA PROGRAMMING SUPPORT ENVIRONMENT
(APSE) EVALUATION METRICS

KATHLEEN D. GRACY
SOFTECH, INC.

Ada Programming Support Environment (APSE) Evaluation Metrics

Kathleen D. Gracy
SofTech, Inc., Dayton, Ohio

A complete Ada Programming Support Environment (APSE) will be a very complex software product and its evaluation and validation will be a very challenging task. This position paper describes this evaluation and validation from two aspects. The first aspect is the use of experience obtained from the JOVIAL J73 Language Control Facility (LCF) activities which performed functions analogous to the planned APSE Evaluation and Validation (E&V) for the much simpler JOVIAL compiler systems. The second aspect is the adaptation of software metrics methodology, developed for the general software environment, to the special needs of the APSE E&V.

The goal of the Ada Programming Support Environment (APSE) Evaluation and Validation (E&V) is to develop techniques and tools which will provide the capability to assess APSEs and to determine conformance of APSEs to the Common APSE Interface Set (CAIS). When these techniques and tools are applied, they must provide software developers and managers with the information needed to select and accept the APSE that most nearly satisfies their requirements.

An important aspect of the APSE Evaluation and Validation is presenting the results in a useful and understandable form. Experience with the JOVIAL LCF activities indicates that this may be the most important aspect. Regardless of how much money is spent, how sophisticated the techniques and tools, how precise the measures, the APSE E&V effort will be for naught if the results are not accepted by the user community. Thus, the first step is to define the user community.

It will be an error if the APSE Evaluation and Validation results are understood by only the small portion of the Ada community that is thoroughly knowledgeable in both Ada and support software technology.

It is important to share the APSE E&V results not just with such experts and seasoned Ada users, but also with organizations just becoming involved with Ada, application analysts, programmers, and managers. Most important is making the results useful to managers for they are the ones who will make the final decision as to the APSE used by their organization.

Throughout the life of the Language Control Facility, the Test Analysis Reports have been to some extent, misunderstood, misused and misinterpreted. Although the goal of the LCF was only to measure conformance of the compiler to the language, it was obvious that the users were interested also in efficiency, code expansion, and general performance. Although this was not the goal of the LCF, users often read into LCF reports an evaluation of such characteristics. For the APSE E&V, the criteria needed by users must be defined from the beginning, and evaluators must make certain that their results are focused correctly on these user needs. For example, one problem with the LCF validation report was in providing percentages of tests passed. The people involved in developing compilers understood that some constructs might require ten tests while others required only one, but that the number of tests had no relationship to the importance of the feature tested. Therefore, a compiler evaluation measure based simply on the percentage of validation tests successfully passed was a misleading indicator of the usefulness or completeness of the compiler. To overcome such a deficiency the tests must be organized into a structure that is based on the functional requirements of the component under test. In turn, these functional requirements may be given weights so that demonstrating compliance with a particular requirement is "worth more" than compliance with a requirement of lesser importance. Many users believe that a relatively high score for conformance is good enough. When significant areas of conformance are not met, weighting should be used to make it clear to the community the seriousness of the deficiency. These weighting factors must be visible to both APSE developers and to users so the factors can be varied to suit the needs of different users or user environments. Considering

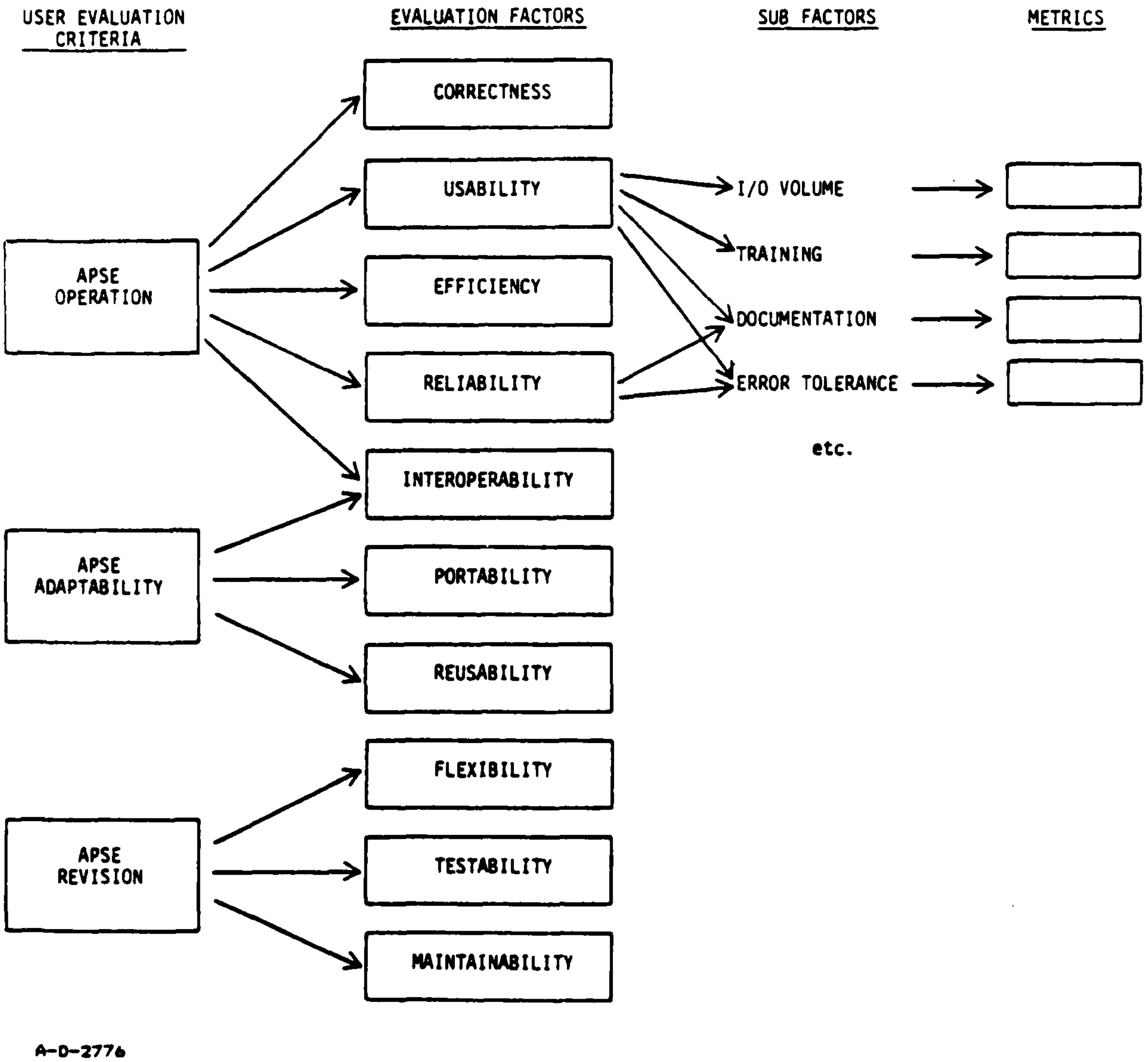
the APSE as a whole, different weighting factors may be required to evaluate the suitability of a particular APSE to the software development situation and to the software maintenance situation.

Another characteristic of the APSE Evaluation and Validation that must prevail, based on JOVIAL LCF experience, is that the evaluation criteria and methods must be known and accepted by the APSE developers as well as the evaluators and the users. This aids in ensuring that the evaluation criteria and test cases are valid since apparent errors in the these criteria and cases will often be discovered by those whose products are being erroneously evaluated. This also increases the objectivity of the evaluation and the acceptance by the developer of even relatively harsh evaluation results. Note that this does not mean that weighting criteria should be a subject for review and criticism by the APSE developer since these weighting criteria are measures of the importance of passing particular tests for particular user environments.

APSE Evaluation and Validation techniques will differ based on whether or not a formal standard exists. Where testing determines conformance to a formal standard, the evaluation can be objective and the results quantitative. Unfortunately, there are many cases, particularly when there is not a formal standard, where only subjective and qualitative evaluation techniques are possible. This situation is analogous to the past efforts in defining software quality attributes and their associated metrics. The APSE Evaluation and Validation effort can benefit greatly from this software quality metric approach.

The overall goal of both the software quality metric and APSE E&V may be stated as the assessment of three major areas: the current behavior/effectiveness of the software, the ease of modifying/enhancing the software, and the ease of transporting and interfacing to the software. Each of these breaks into several factors which must be evaluated

separately. For instance, testability, understandability, and modifiability are all factors that contribute to maintainability. In turn, each of these factors may be broken down into subfactors. Once these subfactors are defined, metrics to assess performance with respect to these subfactors may be devised. The figure below illustrates this structure and is based on the work of McCall and others in software quality metrics.



For evaluations based on the current operational definition, more of the user community's input about evaluation factors will be required. What characteristics are important to users? Although the APSE evaluation task will benefit from user inputs, most people are better at critiquing what has already been done than in coming up with ideas on a blank piece of paper. Therefore, the evaluation factors and associated metrics should be drafted during an initial study and distributed to the user community for comment. At this point we will find out additional evaluation factors the users are interested in and which evaluation factors may not be adequate for the user's needs. Some evaluation factors may require two metrics, one for development and one for maintenance. Some evaluation factors may have an inverse relationship (e.g., the greatest degree of portability may only be achieved by the sacrifice of efficiency and vice versa).

For many evaluation factors or subfactors it will not be possible to define a specific test or tests to determine absolutely whether the APSE has a particular characteristic. There may not be a formal standard specifying required ASPE performance characteristics (i.e., standard deficiency). Even if there is a standard there may not be a method of objectively and quantitatively determining whether the APSE meets the standard (i.e. metric deficiency). The figure on the following page indicates the possible combinations and the resulting evaluation actions for the standard deficiency and metric deficiency situations. The categories are those defined in the APSE E&V Plan.

For example, the Ada compiler component of an APSE has a formal specification (MIL-STD-1815A) indicating which Ada language features must be processed. Using the Ada Compiler Validation Capability (ACVC) as a basis, it is possible to define objective tests to measure whether the

	OBJECTIVE MEASUREMENT TECHNIQUE EXISTS	OBJECTIVE MEASUREMENT TECHNIQUE DOES NOT EXIST
PERFORMANCE STANDARD EXISTS FOR COMPONENT	• VALIDATE COMPLIANCE	• DETERMINE INSTANCES OF NON-COMPLIANCE • EVALUATE SUBJECTIVE FACTORS
	CATEGORY D AND E	CATEGORY C
PERFORMANCE STANDARD DOES NOT EXIST FOR COMPONENT	• MEASURE PERFORMANCE	• EVALUATE SUBJECTIVE FACTORS
	CATEGORY B	CATEGORY A

compiler component correctly recognizes all valid language constructs. Thus, as the above figure indicates one must validate compliance with the performance standard using the objective measurement technique (e.g., ACVC). A user may be interested in the speed of the compiler (e.g., statements compiled per CPU second); for this characteristic there is no defined performance standard. However, it will be possible to define benchmark cases to quantitatively assess compiler speed. Maintainability of the compiler is another characteristic of interest to the user; neither a performance standard nor an objective measurement technique is available for the maintainability characteristic. Thus, subjective evaluation factors based on past software metric accomplishments are needed.

By blending experience from past language control and compiler validation activities with the ongoing work in software metrics, an effective, valid, and usable APSE Evaluation and Validation technique can be developed.

APSE TOOL TAXONOMY

CHARLES HAMMONS
TEXAS INSTRUMENTS, INC.

SECTION 5

POSITION PAPER: APSE Tool Taxonomy

In order for the E&V effort to be a long term success, it is necessary to create a flexible and complete taxonomy of APSE tools. The fundamental ability to classify a tool will be invaluable in identifying validation requirements for tools submitted for inclusion in an APSE. In the sense that the taxonomy provides a framework for definition of basic terms and scope for E&V activities, the careful construction of the taxonomy is crucial to the success of the E&V effort. There are several nontrivial issues that must be addressed in order to create a workable classification system:

1. A more complete elaboration of the overall APSE structure
2. A complete list of tool attributes that have impact upon the capability to evaluate or validate a particular tool
3. Specification of general requirements and allowable variations for each tool class for purposes of evaluation and validation
4. Accomodation of new technology within the general E&V framework: for the taxonomy this generally implies a mechanism for adding classes to the E&V taxonomy as well as addition of new tool attributes that serve the E&V activity

5.1 Overall APSE Structure

The now famous "onion skin" conception of the APSE used to describe its relationship with the MAPSE and KAPSE layers of a software engineering environment is useful as a conceptual model, but is not sufficiently detailed to provide a practical basis for a meaningful classification and validation of tools. The classical APSE conceptual diagram is related to the desired verifiable APSE in the same way that a flowchart is related to a structured program: many flowcharts (APSEs) may be functionally equivalent to the structured program (verifiable APSE), but are not understandable or maintainable (verifiable).

The primitive control structures of structured programming provide a standardization of the elements of algorithm development. Continuing the above analogy, the standardization of interfaces within a more

fully elaborated APSE structure will provide a sound basis for the construction of tools that may be properly evaluated and validated. The elaboration of interfaces along with a more complete elaboration of the structure of an APSE should include:

1. identification of standard interfaces between the gross layers of an APSE, thus specifying allowed interfaces between the tools residing at differing layers of an APSE.

2. identification of allowed interfaces between the tools at a particular layer of an APSE.

3. provision of a mechanism for tightly controlled exceptions to the above standards, generally reserved for new classes of tools not yet invented.

This standardization will greatly contribute to the feasibility of evaluation of an APSE, as communications paths will be constrained. A happy side effect will be the promotion of portable software tools, the software components industry, and the elusive "software bus." This level of standardization probably cannot be achieved in a first generation APSE, as the ALS and the AIE are already well under way, but can be imposed on future development of mature APSEs.

A beginning point for a source of candidate interface standards is the Open Systems Interconnection (OSI) Reference Model for distributed information systems [Day 83]. While this model is generally intended to apply to distributed computing systems as the name implies, the implementation of the common protocols and machine interfaces is sufficiently software intensive to offer a guide for interfacing of software components in an APSE. Further, as this model makes no firm assumptions regarding the physical interconnection or proximity of processing nodes, use of this model as an elaboration of the general APSE structure would support the implementation of an APSE formed from a collection of economical work stations, data base machines, and communications controllers, perhaps spread over a large geographical area. Nevertheless, one could implement such a model within the confines of a typical mainframe computer. Clearly, the above two extremes of implementation strategy would require considerable attention to design of interface implementations with efficiency very high on the priority list.

5.2 APSE Tool Attributes

A taxonomy for APSE tools should promote the evaluation and validation of individual tools to be placed into an APSE as well as the overall APSE itself. A meaningful taxonomy should include as attributes items

from the following two general classifications

1. General categories of tools.

2. Interfaces between tools and layers of the overall environment.

A categorization of APSE tool types has been constructed at the National Bureau of Standards [HOU83]. In summary this work has identified a taxonomy roughly based on the computing paradigm often referred to as "INPUT-PROCESS-OUTPUT." The general classes of tool types has been set as:

1. Management

2. Transformation

3. Static Analysis

4. Dynamic Analysis

Clearly, in many respects this classification can encompass practically all currently known tools that are general in some sense to the development of Ada programs. The taxonomy is further elaborated through the delineation of both input and output characteristics. It should be pointed out that the taxonomy described is not a mutually exclusive partitioning of the "tool space", nor is it likely that any practical taxonomy based upon functionality could form a mutually exclusive classification of tools. For example, an automated requirements processor geared to produce skeleton Ada code in some project data base with traceability "hooks" could be viewed as both a member of the "Transformation" class and the "Management" class. This critique is not meant as an indictment of the work cited, but does lead one to search for more attributes that may be usefully ascribed to APSE tools for the purposes of classification. A pragmatically useful and esthetically pleasing taxonomy of APSE tools should be based upon attributes that may be verified through various methods of testing, whether automated or manual in nature.

The taxonomy outlined in [HOU83] is not geared to express or capture any tool dependencies. While the taxonomy does implicitly recognize that tools may be used in combination to accomplish a particular end ("Pipes" are noted as a possible control input to tools), there is no explicit means to denote the internal dependence of a complex tool written as a system of programs run under control of an APSE executive. Further, for programs that can accomplish their end only with the cooperation of tools that already exist in a validated APSE, the need for such expression is more crucial. Complex integrated tools that are

not easily classified may be dealt with upon a case-by-case basis. examples of such tools are multi-computer hardware test facilities that may provide specialized user interfaces at the APSE level for target system interaction as well as more conventional terminal interaction facilities.

Meaningful specification of functional behavior of software systems is still an object of active research in academia, government, and industry. It does not appear possible at this time to base tool classification for purposes of evaluation and validation upon a functional specification alone. Certain kinds of tools are rather well understood, and may be evaluated in this manner, such as an Ada compiler via a test suite derived from the language specification. A line-oriented editor may be similarly handled, but other tools may not be easily handled in this manner.

The interface requirements of many tools appear to be somewhat simpler to specify in a great many situations. If a tool were specified with respect to its interface requirements as well as its functional behavior, the operational evaluation of that tool may in part be a verification that the tool only uses standardized interfaces for access to services provided by the CAIS or other tools found within an APSE.

Finally, although there is great care taken in the overall conceptual view of an APSE to minimize machine dependencies, there will undoubtedly be tools that are explicitly machine dependent. This is particularly true of tools that provide machine-dependent testing and maintainance support. Such tools are instances of those that may have interfaces to support facilities at the CAIS, MAPSE, or APSE level, as well as interfaces that would be currently based upon the primitive package LOW_LEVEL_IO. It may be that table-driven interfaces may be devised to further hide the machine dependences found in such facilities, but this level of standardization may occur only later in the development of the APSE E&V activity.

5.3 General Requirements

The existence of the taxonomy is not well served unless part of the attributes for a particular class of APSE tool include facilities required to evaluate and validate a candidate tool. There are several issues that must be resolved at this level:

1. What functionality should be required to support tool validation? For example, should all candidate tools support input/output redirection or at least not foreclose use of redirection capabilities within an APSE. One obvious exploitation of input/output redirection would be the creation of files of keystroke sequences for text editor or

word processor validation.

2. Should validated tools be accompanied by a simulator so that mock environments may be built for new tool development or validation if actual use of an existing tool during development/validation is deemed infeasible? Candidate tools that are designed to work in a coordinated way with existing APSE tools may be more easily developed and validated if such simulation facilities are available. This may be particularly helpful for tools that have heavy interaction with one (or more) users or potentially large data bases. The desirability of such an adjunct to validated tools is a matter of technical and economic tradeoffs.

3. What level of parametric adaptability is to be required of APSE tools? By parametric adaptability, we mean the specification of hardware specific or interface specific values by a potentially changeable table or list. Text editors and graphics terminal interfaces are particularly sensitive to variations in the actual target hardware to be used, as will be any general hardware debugging facility. Tools that are not configurable in this or an equivalent manner will likely be so hardware dependant that portability will be severely compromised. Some of these issues may be covered partly in the CAIS specification. However, at this time the CAIS specification is still in a state of flux, and it is likely that some adaptability issues will not be handled at the CAIS level.

5.4 Accomodation of New Technology

The accomodation of technological change in the setting of standards for APSE tools is of fundamental importance to the success of the goals of the E&V task. If explicit mechanisms for evolution and growth of the E&V standards are not incorporated at the beginning, there is some risk that many new tools that incorporate new innovations in hardware and software will be left in "Category A" status [AJPO83, pp 18] subjective evaluation only.

5.5 Conclusions

Most of the above issues, though they require some advances in the state of the art or practice are not purely technical. There are in some respects economic and cultural issues to be resolved. Certain classes of existing tools have achieved a status of respectability and

have established communities of users. The acceptance of existing tools into an APSE may require extensive effort and expense as new standards of development and acceptance are drawn and enforced upon the tool developers. Some tools that are widely used may become less attractive due to the additional code volume of instrumentation required for support of validation. The incorporation of new technology into an APSE is vitally important and the provision for technological change is vital to the success of the E&V effort. In order to successfully cope with the above issues and properly serve the end user of a future APSE, it may be prudent to consider a more evolutionary development of tools within the APSE framework, rather than the creation of full-blown mature tool sets

APPLICATION SPECIFIC - APSE EVALUATION

ASHA KANT
LITTON APPLIED TECHNOLOGY

Applied Technology
Litton

APPLICATION SPECIFIC - APSE EVALUATION

Asha Kant
(Litton Applied Technology)

INTRODUCTION

This paper brings to attention the least addressed area of evaluating the Ada* Programming Support Environments specific to the applications in which they will be employed. It considers the battlefield scenarios and information processing requirements of the future based on Litton Applied Technology's business base and also looks into the need for tools that support the efficiency goals of the Ada language. However, to qualify these tools in an effort to meet the system specification is a future challenge.

GENERAL DISCUSSION

The overall task of evaluation and validation of the Ada programming support environment is very broad, encapsulating a wide area of performance analysis, although the initial goal is to determine the conformance of APSE's to CAIS. APSE-Ada programming support environment is a collection of software engineering tools needed for software engineers to build the software system efficiently (e.g., a subsystem of a mission critical system). Therefore, the evaluation of software engineering tools requires two performance scenarios - one, to meet the goals of software system (subsystem of mission critical systems) and the second is the efficient building of a software system. If we try to prioritize the two performance scenarios, the software system goals would then take precedence. Here, I would like to use an example of a system manufacturer who has a large number of screw types to be tightened. If he were given only one type screwdriver to use, he would not

*Ada is a registered trademark of the Department of Defense, Ada Joint Program Office.

280320-CDS

be able to do the work properly, and maybe not at all, unless he obtains additional type screwdrivers to cover the variation of screw sizes and types in the system. If he has to design and make these screwdrivers at the time when he is supposed to be actually tightening the screws his schedule to deliver the system will be ruined. On the otherhand, if proper tools had been available to him prior to manufacturing start-up, the system requirement could have been met. In software engineering, typically, the software tools are developed in a less timely manner and application requirements are seldom considered in the design of the tools. Software tools are normally generic in nature and are not designed to satisfy the specific performance needs of the software system under development. There exists a need to ensure that adequate software development tools are made available in a timely manner to meet the industry needs for various performance critical applications.

The Ada Programming Support Environment (APSE) must be evaluated in terms of the value these tools bring to the applications in which they will be employed. One such application is Airborne Tactical Electronic Warfare Systems (ATEWS). This application is mission-critical and requires high throughput signal processing featuring high reliability, real-time operation; as well as space, power, and producibility constraints. The accommodation of Ada to these requirements and to the systems produced by Litton, is currently being addressed by Applied Technology in an IR&D project titled "Language Analysis for Electronic Warfare." The results of which will bear directly on the performance requirements for Ada compilers.

The ATEWS of the next decade will perform the traditional radar warning receiver functions (detect, deinterleave, analyze, identify and report threats) in a vastly more complex electromagnetic environment and in concert with other on-board systems. Anticipated pulse densities of up to 10^x pulses per second will levy stringent throughput requirements on signal processors. The increasing complexity of on-board systems is leading to more automation and integration. Data reduction by 'expert' systems will be the basis for automation, but will force ever higher processing throughput requirements.

These will become even more stringent as increasingly complex algorithms and concurrent architectures are required to integrate the sampled environment and report the information to the appropriate on-board systems in real time.

The algorithms and the software required to implement them have traditionally been written in Assembly Language to minimize ATEWS' memory and operation cycle requirements. The application of a high order language to such systems must satisfy the same requirements. Efficiency and speed of operation must be the hallmark of APSE if Ada is to be the basis of ATEWS software.

Let us look at the engineering disciplines involved in building such a product. A product has to go through design, development implementation, test and integration, and life cycle maintenance phases while meeting stringent specification requirements. Each phase must have its own set of tools to make it a success. Software also requires engineering disciplines to make it successful. It's productivity and reliability also depend on the availability of sophisticated tools. Since these tools are inevitably unique, they are long-lead items, and should be turned on long before the product program is turned on. In fact, the tool development process should be continuous as production lessons are learned.

Compliance to the standards is an important issue and productivity improvement is an important goal. However, the ultimate success of a tool lies in how useful it is, how well does it serve its purpose, how well can it build the system. Looking at battlefield senario - software systems similar to ATEWS have always been and still are being done in Assembly Language because tolerances are extremely tight. Reaction time requirements are demanding, space constraints are severe, and the algorithms are heuristic in nature. Can these software systems be written in Ada? The answer at the outset would be "of course" or "why not"?. However, these answers do not quantify or substantiate, and no known examples exist. The first step should be to clearly identify the performance requirements of the Ada compiler and

run-time system for these special applications. There will be challenges yet to be met. Specifically, qualification of these APSE's to meet the system specifications.

CONCLUSION

The past four years have seen the evolution of Ada to the current ANSI-MIL-STD 1983, the formation of AJPO and KIT/KITIA, the development of CAIS and the Ada implementors of ALS for the Army, AIE for the Air Force and ALS/N for the Navy. The formation of Ada Validation Organization to ensure the correct implementation of the standard Ada language has been a step in the right direction. Other efforts have been targeted toward evaluating the adequacy of the language definition from the applications side, and it is the breadth and usefulness of this latter investigation that this paper addresses. In order for Ada to achieve its potential, APSE development must be predicated on the performance and efficiency of various applications, and not just on conformance to CAIS.

BALANCING STANDARDIZATION AND INNOVATION IN THE EVALUATION AND VALIDATION OF ADA PROGRAMMING SUPPORT ENVIRONMENTS

ROBERT J. KIRKPATRICK, JR.
DATA GENERAL

Balancing Standardization and Innovation in the Evaluation and Validation of Ada Programming Support Environments

by

Robert J. Kirkpatrick, Jr
Data General

The Evaluation and Validation (E&V) of Ada Programming Support Environments (APSE's) must be a balancing act between standardization and the need to encourage improvements. Both are absolutely necessary; yet, if they are not carefully balanced, either can spell doom to the overall effort. Fortunately, there are techniques that can be used to ensure a maximum of standardization while retaining the ability to exploit new and improving Software Engineering technology. It is my position that this effort should focus on encouraging the identification and use of those techniques.

There is little doubt in anyone's mind that there needs to be more standardization in Software Engineering. We have long accepted new and substantially different environments each time a new computer or operating system is introduced. Standardization has largely been confined to a single manufacturer where each new effort is advertised as being compatible with the last. Often this advertising is only that and the poor customer is required to pay the costs of upgrading.

If that customer wants some special capability available only on another computer, he is really out of luck. Standardization across manufacturers' machines is non-existent. Recently there have been some gains with the introduction of generic operating systems. UNIX, MS/DOS, and CP/M have gone a long way toward standardization of environments, but they are still a long way from providing a "standard" Software Engineering environment in which to work.

The cost of a great deal of variety in environments is high. most things that are being programmed these days have already been programmed to some extent in other environments. That says that much of the money that is being spent on software these days is being wasted because of the lack of a standard programming environment. How much more productive we could be if we were able to draw on the vast supply of software that has already been produced each time we begin a new sofware project.

As a major manufacturer of software, Data General recognizes that fact and evaluates every new standard that comes out. Even de facto standards perpetrated by competitors with little or no technical merit are considered simply because they are standards. Our customers demand it; we would be out of business without it. No one doubts the need for more standardization in the software business.

On the other side of the scales is innovation. Software Engineering is not a mature field. As fast as we produce a technically exciting software system, someone else produces one that is even more exciting. Systems are obsolete before they hit the market. The worst part is that in order to advance the art, we must do everything within our power to encourage this rapid advance. To institute programs that stifle this innovation, is counter-productive and morally unconscionable.

The computer industry is young and dynamic. Sofware Engineering is younger than most other computer related fields. We cannot expect that the techniques we now use or even those we can envision using will be viable in ten years. We must position ourselves to ride the learning curve of Software Engineering for the foreseeable future. It would be easy to ensure that the APSE effort could not by improperly standardizing upon specific things that will be obsolete in a short time.

Along another dimension of technical innovation is the problem of excluding certain manufacturers from the game by standardizing on something their products cannot support or cannot support well. To believe that these companies have nothing to offer the effort is ludicrous. Big computer companies have no monopoly on good Software Engineering ideas. In fact, many really substantial improvements over the last ten years have come from smaller companies. If we write standards that preclude participation by anyone in the advancement of the art, we will pay the price.

One of the easiest ways to standardize is to find out what is in heavy use today and make that the standard. In several cases, previous standards activities have taken this approach and it has proven to be costly. If we were to do something of that kind now, we would choose VAX/VMS or the IBM 370 or the IBM PC as "standard" or we might write our standard heavily favoring one of those. In five years, those machines and even their architectures may be obsolete. How viable is a standard based on an obsolete machine?

It is therefore incumbent upon those that write the E&V standards for APSE's to be very careful not to inhibit innovation or favor existing systems while at the same time ensuring enough standardization to allow software to be moved between APSE's without change. It is indeed a delicate balancing act; we should not expect a simple solution.

There are a number of mechanical techniques that can help. Being careful not to specify anything about internals is an absolute. Many of the differences between machines can be encapsulated in different internals. If the E&V effort chooses to specify the internals, some systems are not going to be able to support those internals. Rarely, if ever, is it really necessary to specify internals; it is merely expedient. We cannot afford the expedient solution in this case.

Focussing on the interface of a feature rather than how the feature is provided helps, too. As software engineers, we often times have a particular implementation in mind when we propose a feature. Certainly our idea can never be the only way to implement that feature. If we focus the standard on the feature instead of our view of a "correct" implementation, we ensure that the feature on a different system is not constrained out of existence.

Another required principle is to avoid features that are specific to one machine or manufacturer. It is often times difficult for some people to know what those features are because they have dealt so extensively with a particular system. Nothing substitutes for a variety of backgrounds, manufacturers, and opinions present as standards are written.

There are many other simple principles; the ones listed above are just samples. The important thing is that this E&V effort should focus on gathering those simple principles and applying them in its work. Otherwise, its results will be outdated in a few years or counter-productive from the start.

TOWARDS APSE STANDARDIZATION

SUSAN MICKEL
GENERAL ELECTRIC COMPANY

Towards APSE Standardization

Susan Mickel 408/734-4960
General Electric Company
Military and Data Systems Operation
1277 Orleans Drive
Sunnyvale, CA 94086

In the past 1-2 years, as most Ada language issues have been resolved and compilers have become a reality, the focus has shifted to environmental issues. It was realized that a language cannot be truly portable without a standardized environment. One can separate portability problems into two areas:

1. non-standard language features

2. dependence upon environmental resources (e.g. operating systems)

The standardization of the Ada language should eliminate the first type of portability problem. It is now time to address the second type.

The KIT and KITIA were established to develop a single standard definition of the Kernal APSE. This is akin to defining a portable operating system that is so complete that all resources required to develop any software in any application area is already included or can be supplied by writing an Ada package. Quite an ambitious undertaking considering that to date, despite numerous attempts, no truly portable environment has proved successful.

While the CAIS is a noble effort, the goal has not been achieved. The difficulties encountered by the KIT and KITIA were more extensive than originally envisioned. Upon reflection, it is not surprising. Programming languages have been studied for many years. Most of the control and data structures have been established both through theory and use. Still Ada remains controversial. The concept of a software development environment is much newer and has not been studied as throroughly. To date there is no standardization procedure for environments analogous to ANSI language standardization.

This approach of defining a single environment is infeasible, I believe, for the reasons outlined above. A less ambitious, but more viable approach is as follows. First, develop a standard notation for describing the resources provided by an environment; i.e., a systematic method of defining precisely and completely

its capabilities. Such a notation, would facilitate the comparison of different environments, which is currently akin to comparing apples and oranges.

Second, having such a notation and defining some of the most successful existing environments (e.g. UNIX) in terms of this notation, a common set of core resources would doubtless emerge. Searching for such a common set without that standard notation is difficult and imprecise.

Third, instead of seeking to define a single KAPSE, a family of interfaces should be considered. The wide variety of DoD applications makes this necessary. To include all resources required by all applications in every single environment is not feasible. In many cases development is conducted quite differently for different applications: real time versus batch, microprocessor versus main frame. Further, the need to transport systems between application areas (beyond general purpose utilities) has not been established. The goal, therefore, should be the establishment of a family of standard environments.

STANDARDS OF EVALUATION AND VALIDATION FOR
ADA PROGRAMMING SUPPORT ENVIRONMENT TOOLS

JAMES F. PARLIER
GENERAL DYNAMICS

STANDARDS OF EVALUATION AND VALIDATION
FOR
ADA PROGRAMMING SUPPORT ENVIRONMENT TOOLS

James F. Parlier
General Dynamics
Data Systems Division
Western Center
San Diego, California

This paper addresses the issue of Ada Programming Support Environment (APSE) Evaluation and Validation (E&V) concerned with the establishment and enforcement of requirements and standards. The foremost need of an E&V effort, is directly related to customer standards of tool set quality. The acquisition and implementation of tools must be based on the tool's ability to meet or exceed the approved, contracted standard. Standard compliance should be based on a top-down structure of requirements from the most high level requirements established by the Ada Joint Program Office (AJPO) for transportability and operability, to the lowest level of requirements for specific tool life-cycle operations and maintenance. This model for APSE acquisition and implementation will be mirrored completely and consistently in the E&V process.

There have been many studies directed toward the determination of appropriate factors for software tool and tool environment quality. Most of these factors are qualitative rather than quantitative. The top level characteristics typically evaluated are: reliability, testability, usability, efficiency, transportability, maintainability, and interoperability. Of these, the two considered most critical to the APSE are transportability and interoperability. The APSE Interface Team: Public Report, Volume 1, 28 October 1982, defined interoperability as "the degree to which APSEs can exchange data base objects and their relationships in usable form without conversion"; transportability, in the terms used by the APSE Interface Team, compliments this. Transportability of an APSE tool is defined as "the degree to which it can be installed on a different APSE

without reprogramming; the tool must perform with the same functionality in both APSEs".

At all stages of the life cycle of APSE formulation, each tool or tool set should be "verifiable" under these basic terms. The desired product (the requirements and the design) and the actual product (the code) should be represented in a structured, concise, and self-descriptive manner so that they can be efficiently and effectively compared, under the purview of applicable standards. Emphasis on particular quality factors of E&V will vary as the tool or tool set qualifies for implementation into the APSE. However, the specific definition of quality and the "degree" of transportability and interoperability should be specified during the requirements phase of APSE development. It is therefore proposed that APSE acquisition and implementation be viewed within the constructs of a typical systems acquisition model as follows:

REQUIREMENTS DEFINITION	MAKE OR BUY DECISIONS	DESIGN OR PROCUREMENT	DEVELOPMENT AND TEST	IMPLEMENTATION OPERATION MAINTENANCE

Without this structured procedure, standards determination will be virtually hopeless - without standards, Evaluation and Validation would be valueless, and without proper and approved E&V Practices and Procedures, APSE quality and effectiveness will be sacrificed.

For purposes of clarification, the fundamental aspects of a standard will be stated. There are three integrally related considerations; form, fit, and function. Form is defined as the physical characteristics of the tool. Fit is defined as the adaptation of the tool to the desired environment. Function is defined as the performance capabilities of the tool. These characteristics are all directly applicable to APSE E&V. Additionally, these characteristics should be embodied in APSE formulation and APSE E&V standards at each phase of the life-cycle previously delineated. Once approved and in place, determination of the correctness of the final tool or

tool set can be ensured. To emphasize, validation can only be accomplished by verifying each stage of the APSE life cycle. This process must precede any acceptance of the tool or tool set by the customer (i.e. customer certification).

It has been proven time and time again that one of the most prevalent and costly mistakes made on software projects today is to defer the activity of detecting and correcting problems until late in a project. The success of phasing verification throughout the formulation/acquisition cycle depends upon the existence of a clearly defined and stated product at each phase. For example, the Requirements Definition phase product, is a tool set requirements specification. The Development/Test phase ends with the generation of a tested tool set (i.e., the tool set is judged to be technically complete, consistent, and correct, through an examination of the behavior of the tool set by executing it on sample data sets). Consequently, if lower cost and higher quality are the goals, E&V should not be isolated to a single stage in the acquisition process but should be incorporated into each APSE formulation phase.

The following table provides a general overview of the process.

Table K-1. E&V Activities

LIFE CYCLE PHASE	E&V ACTIVITIES
REQUIREMENTS DEFINITION	DETERMINE E&V PLAN DETERMINE ADEQUACY OF REQUIREMENTS GENERATE E&V TECHNIQUES/METHODOLOGIES
MAKE OR BUY DECISION	IMPLEMENT TECHNIQUES/METHODOLOGIES: DETERMINE COST, SCHEDULING, AND PERFORMANCE CRITERIA
DESIGN OR PROCUREMENT	DEVELOPMENT OF E&V PROCEDURES/ SPECIFICATIONS DETERMINE ADEQUACY OF DESIGN
DEVELOPMENT AND TEST	DETERMINE CONSISTENCY WITH DESIGN
IMPLEMENTATION, OPERATION, AND MAINTENANCE	DETERMINE ADEQUACY OF IMPLEMENTATION, OPERATION EFFECTIVENESS/EFFICIENCY, AND SYSTEM MAINTENANCE PERFORMANCE REGRESSION TESTING

Common to all phases are; standards compliance determination, baseline reporting, E&V team status reporting, and requirements traceability and analysis.

Summary

Two principle recommendations are provided in this paper. First, standards detailing the form, fit, and function requirements of an APSE must be established. Following this, standards or specifications outlining E&V involvement in APSE formulation/acquisition could be produced. Second, APSE tool set formulation and/or acquisition should follow a time phased life cycle approach. This will enhance the orderly execution of E&V activities

by providing better visibility, communication, and customer/contractor management control. Though a number of interface issues remain unresolved in regards to the CAIS Version 1.0 of 30 August 1983, like access control, configuration management, and security, APSE E&V will require a concentrated effort on the part of the Ada community to determine and execute standards of excellence for DoD's new and universal computer programming language.

THE IMPLICATIONS OF SOFTWARE TARGETED FOR EMBEDDED COMPUTER SYSTEMS ON THE EVALUATION AND VALIDATION OF ADA PROGRAMMING SUPPORT ENVIRONMENTS

JOHN REDDAN
SYSCON CORPORATION

The Implications of Software Targeted for Embedded Computer Systems on the Evaluation and Validation of Ada* Programming Support Environments.

John Reddan
SYSCON Corporation
3990 Sherman Street
San Diego, California 92110
(619) 296-0085

1. INTRODUCTION

The Ada Programming Support Environment (APSE) is to provide an environment for the design, development, documentation, testing, management, and maintenance of Embedded Computer software, written principally in the Ada programming language. An Embedded Computer System will often execute with an environment foreign to, and more constrained in resources than, the APSE upon which the ECS software is developed. As such, the Evaluation and Validation (E&V) of APSEs should consider the characteristics and tools necessary to develop software on an APSE targeted for the potentially limited Embedded Computer System Environment (ECSE) in its classification scheme.

1.1 Background

Ada was developed with the goal of establishing a single high order language for new DoD ECSs. Early in the development process it was realized that the acceptance and the benefits derived from a common language could be increased substantially by the development of an integrated system of software development and maintenance tools now known as the APSE. Later the Kernel Ada Programming Support Environment (KAPSE) Interface Team (KIT) was established to define a standard set of Kernel interfaces to ensure the interoperability of data and the transportability of tools between conforming APSEs. The Common APSE Interface Set (CAIS) developed by the KIT provides the virtual operating system on which tools run, as well as the minimum set of command, edit and similar functions required to transport tools.

The Evaluation and Validation (E&V) of Ada Programming Support Environments task is being created to develop the techniques and tools which will provide a capability to perform assessment of APSEs and to determine conformance of APSEs to the CAIS. Some of the specific goals of the APSE E&V are to develop requirements for the APSE E&V, and to develop an APSE Evaluation capability.

Some of the steps envisioned to accomplish this (as stated in the E&V Plan [2]) are 1) to identify APSE components, and 2) to identify APSE interfaces and their classsifications. The basis of the identification and classification is a Taxonomy of Tool Features for Ada Programming Support Environments developed by the Institute for Computer Sciences and Technology of the National Bureau of Standards.

The Taxonomy is function/feature oriented, and provides a hierarchal function/feature classification of APSE tools under subjects such as Management, Static Analysis, Dynamic Analysis, and Machine Output. At the feature level, there is a substantial variation in detail, ranging from 11 features per function to 1 feature per function. The current Taxonomy's omission of detail in areas dealing with target dependencies is unfortunate, because many features needed to support targeting of software for ECSEs have been excessively condensed into single features.

2. EMBEDDED COMPUTER SYSTEM ENVIRONMENT REQUIREMENTS

The mission critical software found in Embedded Computer Systems requires a minimal environment which includes a highly efficient, tunable run-time system, primitive file and command handling, and some types of Static and Dynamic Analysis tools. This minimal environment is essential to meeting the strict performance requirements associated with most ECSs.

A critical factor in an ECS is the performance of the Embedded Software. Table K-2 shows some of the ECSE characteristics needed to produce efficient, high performance ECS software. A complete set of characteristics like those shown in Table K-2 can be used to form a set of requirements for the ECSE.

Table K-2. Required Characteristics of ECSE

Feature	Provided By
Position Code/Data Structures in Memory	representation clauses linker
Control maximum memory allocation	pragma controlled pragma memory_size linker
Set maximum number of tasks and active tasks	no use of task objects linker/RTS
Control Overlaying	linker/RTS
Control which Pages/Segments of Memory are Swapped	linker
Control Size/Number of IO Buffers	linker
Insure Exclusion of Unused Library Subprograms	linker

3. EVOLUTION FROM APSE TO ECSE

The APSE itself is not directly required to support any or all of the features required of an ECSE. What is required of Programming Support Environments are facilities to exercise the Target system (including the ECSE) for the development and testing of software for the target system.

Several general approaches exist to solving this problem. They include 1) complete simulation of the target environment including specific and possibly custom devices, 2) cross compilation and downloading of the program to the target computer, and 3) cross compilation and transfer to the target computer via bootable media. Each of the approaches and the numerous variations which exist, possess distinct strengths and weaknesses. A disadvantage of proceeding immediately to the Target System is the more primitive nature of the ECSE (due to efficiency considerations), which often implies fewer testing and debugging tools and resources.

The key to developing Embedded Computer software on an APSE is the existence of appropriate tools to facilitate the targeting of the software. The requirements for these tools are formed in part by the required characteristics of an ECSE, such as those shown in Table K-2. Table K-3 lists some basic tools which might be used in an APSE to target software for an ECSE, depending on the approach selected.

Table K-3. Tools for Transitioning from APSEs to ECSEs

Cross Compiler
Target Machine Loader
Target Machine Instrumenter
Down Loader
Bootable Media Creator
Data File Translator (test file generation and data analysis).

4. CONCLUSION

One of the initial efforts of the E&V Task will be to identify APSE components, interfaces, and their classifications, using as a basis the APSE Taxonomy developed by National Bureau of Standards. The Taxonomy being used as the basis of APSE component classification is primarily function/feature oriented, and does not specifically mention any of these characteristics shown in Table K-2. It also includes many of the tools required for developing software targeted for ECSEs, such as those shown in Table K-3. under the single category OUTPUT/Machine Output/Oject Code of the APSE Taxonomy.

As is readily apparent from the examples shown above, the current Taxonomy does not provide sufficient granularity in the proper areas to address the requirements and tools needed to provide methods for developing software on an APSE targeted for a specific ECSE. The requirements presented by Embedded Computer Systems and their Environments, which are the motivating applications of Ada software in the DoD, should be considered in the formulation of E&V criteria and classifications. The ability to evaluate these areas is currently limited by the APSE Taxonomy deficiences cited above. It is clear that a more refined and consistent Taxonomy is required.

5. REFERENCES

1. Raymond C. Houghton, Jr., "A Taxonomy of of Tool Features for the Ada Programming Support Environment (APSE)", Ada Letters Volume III Number 3, November 1983.

2. "Evaluation and Validation Plan - Version 1.0", Air Force Wright Aeronautical Labortories/Ada Joint Program Office, November 1983.

COMPATIBILITY OF RUN-TIME SUPPORT FOR ADA AND THE CAIS

AMOS M. ROHRER
EG&G, WASC1

COMPATIBILITY OF RUN-TIME SUPPORT FOR

ADA AND THE CAIS

AMOS M. ROHRER

10 FEBRUARY 1984

EG&G, WASCI

MANASSAS, VA 22110

1.0 ADA PROGRAMMING SUPPORT ENVIRONMENTS

The purpose of an Ada Programming Support Environment (APSE) is to support Ada software throughout the life cycle of a computer system. The APSE contains the following components: the data base, the system and user interface, and the toolset. The data base contains all project information. The interface links system users, the system itself, the data base, and the toolset. The toolset includes support software for project development, maintenance, and management.

The APSE has a structure to insure portability of user programs and software tools. The first level of the APSE is the host hardware/software. The second level is the Kernal Ada Program Support Environment (KAPSE) which provides the data base, communication, and run-time support functions to use the APSE. The third level is the Minimal Ada Support Environment (MAPSE) which provides a basic set of tools to produce Ada programs and which can be extended to form a total APSE.

In order for Ada programs and Ada tools to be used in any APSE, each APSE must support a predefined interface for portability. Specification of the interface in terms of Ada packages will make its use available to all Ada programs.

2.0 COMMON APSE INTERFACE SET

In 1982, teams were organized by the Ada Joint Program Office (AJPO) to study transportability of tools and data bases between APSEs. Since then, the KAPSE Interface Team (KIT) led by the Navy and the KAPSE Interface Team Industry and Academia (KITIA) have defined an APSE interface called the Common APSE Interface Set (CAIS) to promote portability of Ada programs and tools.

The CAIS is described in terms of a node model which includes:

Structural Nodes - contain information on relationships between nodes
File Nodes - contain Ada external files
Process Nodes - contain Ada executable programs
Device Nodes - contain I/O device control

Operations on the nodes are defined in terms of Ada packages which form the CAIS.

The particular concern of this position paper involves the implicit run-time support provided by the CAIS for process nodes. There are differences between CAIS processes and Ada tasking which could impact tool usage.

3.0 CAIS PROCESSES AND ADA TASKING

In Ada tasks interact with each other via an entry call. For every ENTRY declaration there is an ACCEPT statement in the body of the task. When the ACCEPT statement is reached, the entry call from another concurrent task is processed in a rendezvous. During the rendezvous, the calling task is suspended

and the ACCEPT statement is executed and parameters are passed. When this critical section of code is finished, the suspended task is reactivated and both continue on.

CAIS process interaction is different from Ada task rendezvous. Processes can send and receive messages over named channels. Processes are suspended until the message is queued on the channel. When a process invokes a new process, it is suspended until the new process finishes. If a process spawns a new process, it is not suspended and continues execution. Processes can signal other processes via psuedo-interrupts.

The differences between CAIS processes and Ada tasks imply that the underlying run-time support will be different also. Implementation of run-time support for both should be compatible so as not to produce conflicting software. This is extremely important on systems where the APSE and Ada applications programs are coexistent. Compatibility of run-time support is important for these systems, especially in the area of APSE run-time monitoring of Ada programs.

4.0 REFERENCES

a. ANSI/MIL-STD-1815A, Reference Manual for the Ada Programming Language, 17 February 1983.

b. Requirements for Ada Programming Support Environments - "STONEMAN", Department of Defense, February 1980.

c. Draft Specification of the Common APSE Interface Set (CAIS), Version 1.1, Ada Joint Program Office, 30 September 1983.

EVALUATION AND VALIDATION ISSUES FOR EMBEDDED COMPUTER SYSTEMS DEVELOPMENT APSES

HELEN E. ROMANOWSKY
ROCKWELL INTERNATIONAL

Evaluation and Validation Issues

for

Embedded Computer Systems Development APSEs

by

Helen E. Romanowsky
Rockwell International
Collins Government Avionics Division

Introduction

The Ada programming language was developed to combat the rising cost of software development, especially the development of embedded computer systems. Developing software for embedded computer applications is hindered by the lack of well-integrated development support tools.

Stoneman [DoD80] defined the requirements for an Ada Programming Support Environment (APSE) to assist in the development of software for embedded systems. In light of the mandatory validation procedures for Ada compilers and since the compiler is but one element of an APSE, it logically follows that an APSE should undergo similar integrity and quality controls. Evaluation and validation (E&V) of the components of an APSE, and as a result the APSE itself, is of integral importance to the quality of software produced with Ada. Therefore, to insure the quality of an APSE and the resultant reliability of the software, it is necessary to establish an accurate set of metrics and standards by which to rate APSE tools.

The E&V effort should be in keeping with the embedded computer development activity for which the Ada language was designed. It is very important that the E&V work be directed towards optimization of the APSEs for use in embedded computer systems development. The E&V work should not be directed towards trying to suit any and all potential areas of application.

This position paper will address two specific issues which deal with the E&V of APSEs for embedded computer systems. The first issue is the development of guidelines to be used by those who want to or have started to build non-DoD sponsored APSE's. The second issue addresses the development of the E&V requirements for tools within an APSE, and more specifically, an avionics oriented APSE.

Non-DoD APSEs

An embedded computer system developer who has begun to use Ada in his work has already made a commitment of sorts to a particular toolset. When that user decided to use Ada, he saw the compiler as the top priority item with respect to which tools

to obtain. Code generators for necessary target machines were of equal importance. Once these have been obtained, then a user may turn to completing his/her "environment."

Exactly how does the rest of the environment completion take place? The user will either develop tools himself or purchase commercially developed tools. Because the "Common APSE Interface Set" (CAIS) was not in existence at the time the compiler was obtained, the user may not have had any plans to make his toolset easily transportable and so was not concerned with whether or not his tools would meet the interface requirements set forth in the CAIS. The E&V efforts can have a large effect on this kind of Ada user.

The role that in-house developed tools and commercially developed tools will play in the APSE E&V effort must be clearly defined. For example, a user has purchased an Ada compiler and linker for use on a VAX. Both of the tools will be tied to the operating system on the VAX. As time goes by, the user decides to obtain additional tools, such as a debugger and configuration management tool, to assist in his software development efforts. These additional tools are also tied in to the operating system of the VAX.

The above scenario illustrates that there is a need for establishing exactly how much of an APSE must undergo E&V. Will the operating system be judged to be an inherent part of the toolset when E&V takes place on that particular environment? The fact that tools rely directly on a particular operating system makes them non-readily transportable to another host machine. This lack of transportability is not compatible with the CAIS. Does this mean that currently existing tools must be modified so that they do not have a dependency directly on the operating system before they can be considered as members of a valid toolset?

Within the E&V requirements, there must be specific guidelines written which will define the position of these vendor or in-house supplied components of an environment with respect to their influence on the evaluation and validation of a set of programming support tools. If a user has already committed to using a particular set of tools, then he has already made an investment in those tools.

The most obvious investment is in dollars because the tools were either bought or developed in-house. This dollar cost is something which should not be dismissed lightly. A company should not be forced to stop using tools already in-house just because they may not have been DoD sponsored. The commitment to use Ada in a company that currently has a toolset had to come before a

full APSE was developed and before the CAIS document existed. This type of user should be given fair consideration in the E&V requirements and standards.

Perhaps more important than a dollar cost, there has also been an investment in the tools from the standpoint of the programmers who have been using those tools to develop software. These people have already overcome the learning curve with respect to how the tools work and are familiar with their operation. The shortcomings of the tools have probably been realized and work-arounds developed. The advantages of the tools have also been seen and worked into each individual's programming habits. These people have come to expect certain levels of performance from their tools. This investment should definitely be taken into account by the E&V plan.

Another side to this issue deals with those users who currently have just an Ada compiler, but want to build and/or buy a set of software development tools. Even though the E&V effort is directly aimed at DoD sponsored environment implementations, an effort to construct a non-DoD sponsored APSE should be specifically accommodated. The final E&V plans should include a set of requirements and/or recommendations for how this kind of an implementer should proceed and to what standards he must conform.

The programming support environment which a user builds or acquires is to assist in the development of software. This software should be of high quality and should be developed in a manner which can facilitate the attainment of this quality goal. Each APSE site will no doubt have its own ideas about how software should be developed and will want to obtain tools which reflect this methodology. Therefore, the harmonious nature of the APSE becomes an issue. The E&V plan should take into account how different software development methods may need to use different tools and thus propose guidelines as to which methods are best supported by which particular tool characteristics.

Evaluation and Validation of Avionics Oriented APSEs

This issue deals with the requirements for tools in a specific area of embedded systems applications, that of avionics. Stoneman [DoD80] discusses what tools should be in an APSE, but exactly how much of the actual specification of tools should be done in an E&V effort? How much should the area of application affect the degree of generality which E&V will allow? Once the toolset has been outlined within the E&V effort, does a specific

application such as avionics need any additional clarification, restrictions, or enhancements to its APSE toolset?

Evaluation is defined in the Evaluation and Validation Plan as assessing the quality of APSE components [AJPO83]. Will this quality assessment include a test for correctness of a tool? How deep of a role will the E&V take with respect to determining exactly how correct the tools are that will be used in avionics software development. For example, if a simulator must be used in order to develop avionics software for a specific contract, does the simulator itself come under the evaluation and validation effort?

Software which is developed for avionics embedded systems applications is oftentimes done via the use of cross-compilers. This introduces another element to consider when setting up evaluation and validation plans. When validation of a compiler takes place, it includes both executable and non-executable tests. When validation of a cross-compiler takes place will it have to include execution of tests upon the target processor? If so, will this mean that there will be an addition of target-based testing hardware which must be part of the APSE? Development of software for embedded computer systems inherently means that this issue will have to be dealt with during the evaluation and validation process.

The quality of APSE components must be judged in conjunction with the degree of reliability which these tools give to the software which they are used to develop. Again, with a simulator as an example, the reliability of the software depends in part on how correct the simulator itself is. Problems will occur, however, when trying to evaluate or validate tools such as a simulator which are based on a company proprietary processor architecture. It would be a large undertaking to make up evaluation or validation tests for these tools because they are not of a general nature. Those designing the standards or metrics would need to understand each processor's architecture. A policy needs to be determined as to what extent these kinds of tools will be affected by APSE E&V.

Conclusion

The initial E&V plan should provide for a toolset which will assist in the development of embedded systems software and still provide a pathway for expansion. To assist in the development for embedded systems there is a need for the E&V effort to address the questions of performance and capacity. For some

embedded systems, the performance of the tools, especially the compiler, has a marked effect on how well the development process progresses. Any of the realized savings on life cycle costs may be facilitated or diminished based on the good or bad performance of a tool. The question of capacity also has an influence on life cycle costs. If the tool cannot handle a particular system or program size, then its overall benefits need to be realistically evaluated.

The undertaking of the E&V task to "develop techniques and tools to provide a capability to perform assessment of APSEs and to determine conformance of APSEs to the CAIS" [AJPO83] is not going to be an easy one. In spite of its difficulty, the APSE E&V task is one which should prove beneficial to the embedded computer systems community at large. Although there has been controversy over whether or not the validation suite for the Ada compilers has been effective enough to help those who do produce software for embedded computers, there is a chance to avoid this kind of problem by making certain that the APSE E&V plan specifically provides for the needs of embedded computer system developers.

REFERENCES

[AJPO83] ---------, "Evaluation and Validation (E&V) Plan, Version 1.0," Ada Joint Program Office, 30 November 1983.

[Buxt80] Buxton, J. N. & Druffel, L. E., "Requirements for an Ada Programming Support Environment: Rationale for Stoneman," IEEE COMPSAC 80, October 1980, pp. 66-72.

[DoD80] ---------, "Requirements for Ada Programming Support Environments, STONEMAN," U. S. Department of Defense, February 1980.

[Houg83] Houghton, R., "A Taxonomy of Tool Features for the Ada Programming Support Environment (APSE)," Ada Letters, November/December 1983, pp. 69-78.

[Stan82] Standish, T., "The Importance of Ada Programming Support Environments," AFIPS Conference Proceedings, 1982 National Computer Conference, June 1982, pp. 333-339.

INCREASING APSE CAPABILITIES IMPACT ON E&V

ANDRES RUDMIK
GTE COMMUNICATION SYSTEMS

INCREASING APSE CAPABILITIES IMPACT ON E&V

Andres Rudmik

GTE Communication Systems R&D
2500 W Utopia Rd.
Phoenix, Arizona 85027

INTRODUCTION

This paper identifies a basis for APSE E&V and shows how the E&V effort will increase with increasing APSE complexity. APSEs can range in complexity from a simple single-user, single-program development environment to a large scale multi-user, distributed environment. Certainly it seems intuitive that it will take more effort to evaluate and validate the more complex APSEs. This paper suggests some basic APSE E&V criteria and shows how this criteria might be applied to APSEs of varing complexity.

The arguments presented in this paper will be illustrated in terms of a project in which we are developing a Distributed Software engineering Control Process[1] (DCP). The DCP is a portable distributed Ada[2] programming support environment that provides centralized project management and control facilities integrated with an off-the-shelf Ada compiler and associated development tools. Even though the DCP was not developed on top of the Common APSE Interface Set (CAIS), we will use the DCP to illustrate how some of the CAIS interface standards may be validated.

DCP A CASE STUDY

The DCP has a layered architecture as illustrated in Figure K-3. The heart of the DCP is a database that maintains and manages information about the DCP users, the objects under development, and the development process. The DCP database can be viewed as a single centrally controlled database containing all the system wide application information and documentation. All the database accesses will be controlled to ensure consistency of data so that a DCP user can obtain complete, accurate, and current descriptions of applications, all their parts and the relationships between their parts.

---

[1] The DCP development is funded by the WIS JPM Technology Directorate under contract MDA-903-83-C-0202.

[2] Ada is a registered trademark of the U.S. Government (Ada Joint Program Office).

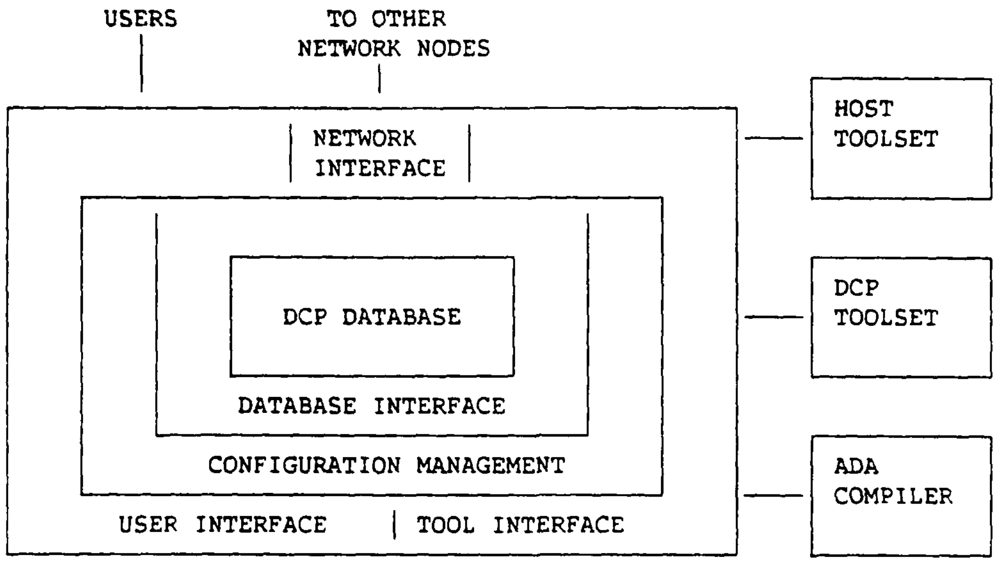


Figure K-3. Layered Architecture of DCP

The database maintains a directory which allows users and tools to refer to DCP objects in a host transparent manner by maintaining a mapping between logical and physical file names. The DCP user communicates with the DCP using logical file names and invokes DCP tools using these names. The tool interface is responsible for converting these logical names into physical file names and then directing the tools to operate on these files. This approach supports a distributed development environment where the user need not be concerned with where the files are stored and how they are accessed.

Surrounding the database, there is a portable database interface that allows the DCP to be ported to other hosts where there may be different but compatible databases. The database interface will provide standard access operations to all the DCP facilities and tools, and a standard database query interface.

On top of the database, we have built a configuration management system to ensure that all applications and database designs developed using the DCP are maintained in a consistent form. The design of the configuration management system uses the database to manage information about the constituent components of a document or a program. Configuration management tools will allow users to define a configuration, to add components, fetch components, store components, compile programs etc. These tools contain logic to ensure that the DCP user is performing a valid operation and is not violating the project development procedures and standards.

The outer layer of the DCP provides a portable interface to the DCP users and the DCP toolset. The user interface supports the use of Ada as a command language, a full screen menu system, a help facility, and an on-line documentation capability. Some of the tools are host dependent and will therefore be different on each host, in which case, the tool interface would be modified to accomodate these tools. The DCP user interface to the tool would remain the same with the only differences occuring when the user is interacting directly with the tool. Even these difference can be eliminated in time as portable tools are developed in Ada. A network interface will allow other DCP development hosts to be interconnected to provide a distributed development capability. The distributed properties of the DCP will be fully realized when the distributed database capability is available.

Our approach to building the DCP is to maximize the use of off-the-shelf tools and concentrate our efforts on building an integrated environment by developing virtual interfaces between the tools, the user, and the DCP database. As part of the DCP contract we are to demonstrate the portability of the DCP by porting the DCP to a different host. This exercise will provide an opportunity to evaluate the portability of the DCP interfaces and the toolset.

DCP CONFORMANCE TO ADA

It is assumed that the DCP will use a validated Ada compiler when one becomes available on our development host. The validation of the compiler in a simplified environment may be inadequate when the same compiler is used in a sophisticated environment such as the DCP. The Ada Compiler Validation Capability (ACVC) is primarily oriented toward the validation of a compilers comformance to the Ada LRM. This validation procedure does not take into consideration environmental issues which will also affect the correctness of the Ada programs. For example, multiple developers working on the same program will introduce time dependencies into the APSE's view of the developing program. It will be the responsibility of the APSE tools to ensure that the program produced is consistent and correct.

An APSE such as the DCP must support the entire software life-cycle ensuring that the entire product consisting of requirements, design, implementation, and test plans are developed and maintained in a consistent manner. The DCP has tools to perform impact analysis on changes to determine the program components that are affected by the change. Furthermore, packages can be shared between programs such that changes in one program may affect another program, in which case the APSE must carefully manage the sharing of program parts. These arguments suggest that one criteria for APSE E&V is its ability to maintain a consistent and correct representations of Ada programs.

The Ada Compiler is only one of many tools that will be used to support the development of Ada programs. The APSE validation procedures must address the issues of validating all tools that process Ada source, that present information using Ada syntax, or generate Ada source. One of the problems

here is that the ACVC may be inappropriate for this validation. For example, consider a tool that allows the developer to query an Ada program library to display Ada data structures. A conceivable validation procedure would be to show that all data structures displayed conform to the LRM and accurately represent the original data structures. Other tools that generate Ada source text should also be validated to show that the generated code conforms to the Ada LRM.

If the criteria that APSEs must maintain a consistent and correct view of Ada programs is to be used in the APSE E&V effort, then it becomes apparent that as the ways in which Ada text can be created, managed and dispayed increases in complexity so will the APSE E&V. Even though this aspect of the APSE E&V is related to the ACVC, in most cases, the ACVC facilities cannot be used directly because of the special nature of the APSE tools (ie. they do not necessarily process Ada source as input).

DCP PORTABILITY VALIDATION

One of the goals of the DCP project is to produce a portable APSE where rehosting the DCP does not require modifications of the DCP toolset, preserves the user interface, and allows the project database to be distributed between the hosts. Our goal was to port the DCP by just modifying the interface implementation while preserving the interface specification. Figure K-4 illustrates how one could validate the portability of the DCP and show that the DCP components conform to the virtual interfaces in the different host environments.

```
                 --- Evaluate DCP specification      ===> Evaluation of
                |    against STONEMAN.                    DCP specification
                |
                 --- Evaluate DCP validation tests
                |    and procedures against DCP      ===> Evaluation of
                |    specifications                       validation plan
                V
1)  (DCP) + (DCP validation tests and procedures)
    ---------------------------------------------  ==> Validation
        DCP Host Virtual Interface (VAX/VMS)             Metrics


2)  (DCP) + (DCP validation tests and procedures)
    ---------------------------------------------  ==> Validation
        DCP Host Virtual Interface (Other Host)          Metrics
```

Figure K-4. E&V of DCP Portability

The E&V of the DCP portability could be done in stages as illustrated in Figure 2. The first stage is to evaluate the DCP on the existing host showing that the DCP specifications conform to the Stoneman requirements. Next, the DCP validation tests and procedures must be evaluated to show their adequacy in evaluating the DCP components and the entire DCP system. Finally, the validation tests and procedures would be applied to the DCP on the VAX development host. At this point we can derive validation metrics to show the degree of conformance of DCP to the validation tests. These metrics would include a measure of successful tests and perhaps a measure of the test coverage on the DCP system.

In order to validate the DCP portability requirement we would select a different DCP host and reimplement the DCP interface set on the new host. The validation of the DCP on this host would then involve the use of the validation tests and procedures used on the development host and a comparison of the validation metrics. We would expect a high degree of correlation between these metrics on both hosts to demonstrate portability.

In addition to the E&V criteria discussed in this paper we may also want to consider the following:

1. Measure of project database transportability and database tool interoperability.

2. Evaluation of DCP performance on different hosts.

3. Evaluation of DCP robustness.

4. Evaluation of DCP user friendliness.

5. Validation of DCP toolset conformance to the virtual interfaces.

6. Validation of protocols between DCP tools. When is this necessary? How would this be validated?

7. Evaluation and validation of user interfaces.

8. What about tools that do not use the CAIS?

9. What about tools that are not supportable by the CAIS?

10. Criteria for evaluating and validating Ada Libraries. Support for package reusability.

11. Evaluation of development methodologies supported by APSE.

12. Evaluation of APSE support for project management, control, and security.

ADA PROGRAMMING SUPPORT ENVIRONMENT EVALUATION AND VALIDATION (APSE E&V) USER INTERFACES AND METHODOLOGY COMPATIBILITY: TWO FORGOTTEN ISSUES

RAYMOND E. SANDBORGH
MICHAEL J. MEIRINK
SPERRY CORPORATION

Position Paper: Ada Programming Support Environment Evaluation and Validation (APSE E&V) User Interfaces & Methodology Compatibility: Two Forgotten Issues.

Sperry Corporation
Raymond E. Sandborgh, Staff Consultant
Michael J. Meirink, Supervising Programmer

One difficult aspect of developing a position on APSE E&V is selecting an issue to discuss as there are so many important ones. Table K-III is a partial list of issues. Of all these issues, two have been selected. These issues were choosen because, 1) it is difficult to see how to approach a solution to the issue, 2) the issues are of a broad general nature, 3) each issue is important to the E&V process, and 4) we have developed some experience in the issue choosen. All of the issues in Table 1 are important, we trust, as a result, that others with different prospectives will choose to take positions on them. The two we have selected are:

1. The problems of describing the user interface in sufficient detail and scope to perform systematic comparisons of APSEs.

2. The degree to which an APSE supports or precludes a specific genera of methodology, to wit:

   a. Rigorous
   b. Formal
   c. Procedural
   d. Heuristic
   e. Artistic
   f. Ad-Hoc

In the following section both issues will be described in more detail, an example of a specific instance will be presented and a potential approach will be overviewed.

The User Interface

According to STONEMAN, one of the goals of an APSE is the creation of a virtual environment that would allow a programmer to use the same programming language, Ada and the same commands to access and control computer resources regardless of hardware. A parallel capability would also be made available to managers, a constant means of obtaining reports, managing a configuration, etc. for any site or project would be available. The idea was to reduce the amount of information a person needed to operate the Software Engineering Environment over the long run, with a single interface. Now, there is no agreement that a single user interface is necessary or even desirable. None the less, the question is, "What should the APSE user interface look like? (Perhaps, the phrase ... and sound like ...) should be added, as tone and even speech generators are economically reasonable.

Figure 1 provides a basic structuring of the user interface domain as a 2x2xn matrix. This matrix has Target Group as one dimension and Display Variable Group as the other. The Target Group Dimension is divided into Programmer and Manager, while the Display Variable Group is divided into format[1] and content[2]. The third dimension of the matrix is APSE Instances. The matrix aids in the development of questions such as the following:

- o Should any of the APSE content be the same for programmers and managers? What is it? Is this content actually the same?
- o Should the user have control over the format? (For example, be able to switch from menu to query to command structure.)
- o What are instance differences?
- o What is a difference or how close is close enough?

Let's look at one aspect of the problem.

For the greatest productivity and lowest error rate, should the format for an APSE be 1) Menu Driven, 2) Query Driven, 3) Command Driven, or 4) User Selectable among 1, 2, and 3? Should managers and programmers have the same format if choice 4 is not selected? It the same format appropriate for all tasks?

In order to answer these and similar questions, in even a roughly optimal manner, it is necessary to consider the user, the task, and capabilities of the computer system (hardware and software). Considering just the user at this time we need a way to describe how sophisticated a user is likely to be as an APSE user. A basic model for this user characteristic is available from learning psychology. This is the skill acquisition model, the stages are:

- o Rote - The person can follow correct instructions.
- o Novice - The person can work with several specific, isolated capabilities within the context of the system.
- o Intermediate - The person can perform identifiable portions of the task readily, within the context of the system.
- o Advanced - The person can use the systems capabilities in novel ways to solve problems.
- o Expert - The person can extend the use of the system into new domains.

[1] Format is the appearance of the display and the means to control the display's appearance, i.e., menu vs command format.

[2] Content is the "What does the system do?, it is the system's function and performance.

1)	How will an APSE be judged as "complete" and "mature"?
2)	How will APSE differences impact acceptance by both users and validators? (i.e., ALS vs AIE vs ALS/N etc.)
3)	What type of validation is appropriate for which parts or features of the APSE, a validation suite vs some form of utility assessment?
4)	To what degree should the APSE be methodology level independent or specific?
5)	How will the user interface be described with any degree of precision and formalism.
6)	How will measurement interaction issues such as, one group has experience with a similar programming support environment and another group doesn't, or the act of measuring will change the group's performance.
7)	Is the system attack proof or secure? How will you tell?
8)	How do you separate KAPSE, MAPSE and APSE evaluation?
9)	How do you compare APSE's to each other?
10)	How will resource performance be measured?
11)	Will such items as nominal and maximum sizes be evaluated, for example, max file size, number and capabilities of terminals?
12)	How will a CAIS Validation Suite be created?
13)	What information should be subject to APSE validation concern: only product information or should nonbaseline results, planning data, people/computer resources be included?
14)	The problems of describing the user interface in sufficient detail and scope to perform systematic comparisons of APSEs.
15)	The degree to which an APSE supports or precludes a specific genera of methodology, to wit: a. Rigorous b. Formal c. Procedural d. Heuristic e. Artistic f. Ad-Hoc
16)	How valid and reliable is the work of the evaluator/validator?

Table K-4. Issues in APSE E&V

Figure K-5 provides a basic structuring of the user interface domain as a 2x2xn matrix. This matrix has Target Group as one dimension and Display Variable Group as the other. The Target Group Dimension is divided into Programmer and Manager, while the Display Variable Group is divided into format[1] and content[2]. The third dimension of the matrix is APSE Instances. The matrix aids in the development of questions such as the following:

- Should any of the APSE content be the same for programmers and managers? What is it? Is this content actually the same?

- Should the user have control over the format? (For example, be able to switch from menu to query to command structure.)

- What are instance differences?

- What is a difference or how close is close enough?

Let's look at one aspect of the problem.

For the greatest productivity and lowest error rate should the format for an APSE be 1) Menu Driven, 2) Query Driven, 3) Command Driven, or 4) User Selectable among 1, 2, and 3? Should managers and programmers have the same format, if choice 4 is not selected? It the same format appropriate for all tasks?

In order to answer these and similar questions, in even a roughly optimal manner, it is necessary to consider the user, the task and capabilities of the computer system (hardware and software). Considering just the user at this time we need a way to describe how sophisticated a user is likely to be as an APSE user. A basic model for this user characteristic is available from learning psychology. This is the skill acquisition model, the stages are:

- Rote - The person can follow correct instructions.

- Novice - The person can work with several specific, isolated capabilities within the context of the system.

- Intermediate - The person can perform identifiable portions of the task readily, within the context of the system.

- Advanced - The person can use the systems capabilities in novel ways to solve problems.

- Expert - The person can extend the use of the system into new domains.

[1] Format is the appearance of the display and the means to control the display's appearance, i.e., menu vs command format.

[2] Content is the "What does the system do?, it is the system's function and performance.

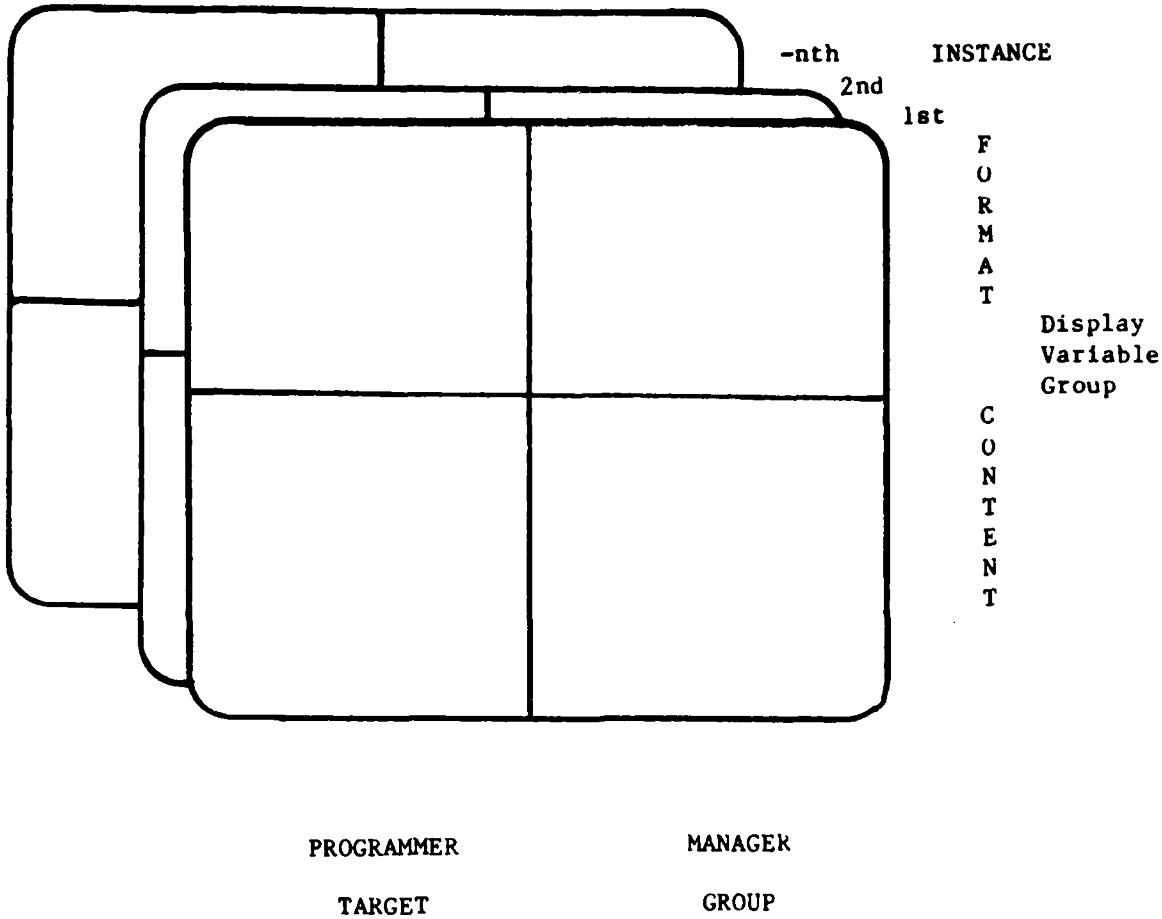


Figure K-5. User Interface Domain Structure for APSE E&V

The factors in determining into which category a user belongs are interaction mode, object generalizability, operation generalizability, preplanning, chunk size, and chunking approach.

These categories are further defined as follows:

- o Mode of Interaction
  - passive
  - reactive
  - active
- o Object Generalizability
  - concrete
  - superordinate
  - extended
- o Operation Generalizability
  - independent
  - combinatorial
  - extended
- o Preplanning
  - informal
  - formal
  - predictive
- o Chunk Size
  - character
  - item
  - statement
  - procedure
- o Chunk Approach
  - passive
  - informal
  - formal

These variables are used to define a user state, initially this is all zeros but ones appear as competency develops. The transition from rote to novice to intermediate to advanced to expert is a series of thresholds. The fit of interface to a user group can be evaluated using this approach. On Figure K-5 this approach provices an evaluative method for one variable value - program format. Three other evalvative methods need to be developed.

The point is, the most explicit, replicable and cost effective models need to be used and are in some part available for evaluation and validation of the human interface in APSEs.

<u>Software Engineering Methodology and the APSE</u>

According to STONEMAN, level 3 APSEs are:

> "Ada Program Support Environments (APSEs) which are constructed by extensions of the MAPSE to provide fuller support of particular applications or methodologies."

However, neither level 1, KAPSE nor level 2, MAPSE definitions have any methodological content. For these portions of the APSE, it is clear that methodology must not be prevented. Yet, no active support is required. Some form of evaluation, perhaps a check list for each of several methodologies, with and without tool support, would suffice from a methodological perspective.

We have found it useful to think of "types of methodologies" by which we mean the way a methodology varies on key characteristics such as:

- The problem domain or environment of the methodology
  - the body of content or application the methodology must address
  - the set of audiences skilled in interacting with the methodology
  - the outcomes to be achieved
- How competent the persons working with the methodology are in the use of the methodology.
- How easily the methodology is evaluated in a public manner
- How clear the information requirements are, both in a particular activity and between two or more activities.

Using these variables, the following methodology types have proved useful.

1) Ad Hoc Methodologies - This is a trial and error approach. To some degree work cannot be planned in advance as it is based upon immediately prior results.

2) Artistic Methodologies - Depend upon the skill of a master programmer. The broad routines can be described, but actual execution is dependent almost totally upon skill level of the staff.

3) Heuristic Methodologies - Here the major ways of approaching a problem and a variety of specific techniques are well articulated. Closing criteria are missing or if present are partly or wholly arbitrary.

4) Procedural - The process is explicitly articulated. It identifies the major steps (perhaps substeps also), the information handled at each step, key measures and metrics, and completion and quality criteria. The intent is to influence the order in which decisions are made and to improve visibility and control over the process.

5) Formal Methodologies - These methodologies use notations with a definite syntax and a partially defined semantics but the meaning applied to the same is somewhat arbitrary. The syntax is itself formally defined, i.e., by a Bachus - Naur representation, or syntax charts.

6) Rigorous Methodologies - Are those formal methodologies which have precise conceptual meaning, due to a mathematical base such as Petri Nets or Finite State Machines, and which have sufficient operational definitions to allow emperical validation of constructs and their representations.

Furthermore, from a user's viewpoint, all of these software development methodology types have these key components:

Notations: - language used to describe the system to be developed.
Methods: - techniques to develop and to determine the development's compliance to criteria.
Tools - automated support for handling notations and for encouraging/enforcing methods.
Procedures - written description of the proper use of notations, methods, and tools.

There are several motivations for raising this issue:

(1) STONEMAN expects to support methodologies: "Ada Program Support Environment (APSEs) which are constructed by extensions of the MAPSE to provide fuller support of particular applications or methodologies." Concern for the disciplined use of the APSE motivated the METHODMAN effort.

(2) Organizations will strive to improve visibility and control over the software development process. That is, the entire process must not be totally ad hoc or artistic. The process must be appropriate to the task. Each of the methodology types have an appropriate use--innovation (artistry must not be stifled), some processes can be mechanized (procedural, formal, rigorous). Productivity gains can be achieved by improving tool capability and increasing tool use.

(3) Interest in the front end of the software life cycle is high. The majority of errors occur then; the later they are detected, the more costly they are to repair. Many have developed or are developing tools to support an "Ada Program Design Language". A major toolset under the auspices of the Joint Services Software Engineering Environment (JSSEE) Committee is the Distributed Computing Design System (DCDS). The Army CECOM has contracted the development of methodologies whose notation for requirements specification and design is pure Ada (Ada Formulation Methods Study). These methodologies promote the notion of merging the process of conceiving (creating) the design with the process of recording it. The latter two are examples of rigorous approaches. A key issue which emerges is what information beyond Ada is needed to describe design? Today, contractors must look beyond tools written in Ada or conforming to a CAIS interface.

From a buyer's viewpoint, the key questions are:

(1) Can this APSE support my methodology?

(2) What tools are available to support a methodology?

(3) Is this methodology and its supporting toolset applicable to my problem (or project)?

(4) What are the workproducts of the methodology?

It is important to recognize that tools may be integrated with respect to other tools (i.e., program callable interfaces to the database or inter-tool data), with respect to the user interface, or with respect to a methodology. Thus, tools used in the context of a methodology may comprise an integrated set regardless of other aspects of integration.

The crucial question is "What is needed to evaluate the suitability of an APSE to one or more methodologies?" APSE support for methodologies cannot be meaningfully evaluated by microscopic analysis of tools and their interfaces. There are, however, several reasonable approaches which can be taken:

(1) Identify information items and questions which could serve as a basis to begin evaluation;

(2) Develop benchmark problems for industry or academia to use to exercise methodologies and their toolsets;

(3) Issue guidelines for conducting data-gathering exercises. The guidelines should address controls, transition between tools and methodology tasks, errors, tool and people performance, and analysis of results.

The following strategies and criteria are recommended for evaluation:

(1) A tool (or toolset) said to support a methodology should supply the following information:

   (a) Description of the methods suggested via an explanation of the analysis techniques, underlying formal foundation, and rationale.

   (b) Definition of the notation supported--
      (i) Formal Definition (syntax and semantics)
      (ii) Cite and explain Ada compatibility
         a. Notation is a subset of Ada--identify exclusions;
         b. Notation is a subset of Ada plus extensions--identify exclusions and extensions;
         c. Notation is a subset of Ada--identify extensions;
         d. Notation is not Ada-based--if applicable, explain mapping to Ada.

   (c) Describe toolset capability--inputs, outputs, functions, manuals, etc.

   (d) Provide procedures (layer by detail)--
      (i) Overview:
      (ii) Heuristic (major process steps);
      (iii) Procedural.

(2) To evaluate the productivity of using an APSE environment according to some methodology, E&V should consider:

(a) Methodology and APSE support for controlling the development;

(b) Definition of errors:

(c) Size and complexity of the application problem;

(d) Level of training;

(e) Data-gathering and analysis techniques;

(f) Number of participants.

It is worth noting that METHODMAN passes the eight questions for conducting experiments even though the guidelines chosen appear to be consistently flawed.

(3) To evaluate the extensibility and flexibility of an APSE in supporting multiple methodologies, E&V should consider the following for multiple projects or methodologies:

(a) How easily is the notation tuned?

(b) How easily can the decision rules encouraged by the toolset be tuned?

(c) How easily are the procedures altered?

(4) To assess the ease of transitioning across tasks or phases, E&V should consider:

(a) Information passed (to next phase);

(b) Information saved (as history, hidden);

(c) Information sunk (no longer needed;

(d) Information lost (should hae been passed).

(e) Activities (actions) interacting among phases.

This entire discussion is rather abstract, due not only to space limitations but also to the lack of a body of knowledge about this topic. For example, it would be useful to have experimental data that compared:

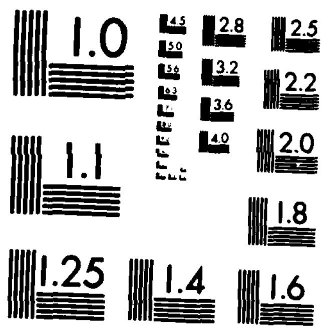
- o 2 APSES (e.g., Rohm vs. TeleSoft)
- o 2 Problems (e.g., C&C vs. ATC)
- o 2 Methodologies (e.g., "ARMY CECOM Ada Integrated Methodology" vs. "DCDS").

Even careful work on the design of such an experiment, with the associated "thought experiment" work and directed literature research such a design entails would produce useful insight. A final comment is in order. None of this discussion directly addresses evaluating (or validating) conformance to a CAIS.

AD-A153 609 EVALUATION AND VALIDATION (E&V) TEAM PUBLIC REPORT 6/6
VOLUME 1(U) AIR FORCE WRIGHT AERONAUTICAL LABS
WRIGHT-PATTERSON AFB OH V L CASTOR 30 NOV 84

UNCLASSIFIED AFWAL-TR-85-1016-VOL-1 F/G 14/2 NL

END
FILMED
DTIC



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

The reason for this is simple. A CAIS is one solution to the problem of portable software. Others are known to exist (UNIX, for example). As a result it is not critical to have a CAIS to achieve the productivity and quality increases which are the main thrust of the Ada effort. Of course, it is necessary to have a CAIS in order to validate conformance to a CAIS. But the practical issue in APSE E&V is which APSE, if any, is most likely to be best suited to my project (or procurement) objectives and the type of methodology that is going to be used by my performing agency (or contractor).

COMPREHENSIVE SOFTWARE DEVELOPMENT ENVIRONMENTS

PAUL SCHEFFER
MARTIN MARIETTA DENVER AEROSPACE

APSE Evaluation & Validation Workshop

Position Paper

Comprehensive Software Development Environments

March 1984

Paul Scheffer
Computer Systems Technology
M/S 0421

Martin Marietta Denver Aerospace
P.O. Box 179 Denver, Colorado

COMPREHENSIVE SOFTWARE DEVELOPMENT ENVIRONMENT

Martin Marietta Denver Aerospace initiated intensive research in software engineering methods, techniques, and tools in 1976. Throughout the evolution of our current capability we have studied individual tools, methods, procedure integration techniques, automated documentation schemes, and software management problems and solutions. Our hard earned understanding of the criticality of the man-machine interface component of successful software tools, and more recently, viable tool systems was not achieved without discovering the pitfalls to be avoided. Early developments of large tool schemes which combined complex model based specification methods with elaborate command languages, not only resulted in burdensome resource consumption from data manipulation and structuring, but produced packages of such complexity that user acceptance was precluded. Training times from such approaches are so extensive that staff acceptance and simple usability become overwhelming managerial and educational problems. An obvious counteraction to the development of large, complex tool systems is an approach wherein small, definitive scope tools of limited capability attack specific life-cycle problems. A great intuitive appeal is associated with such schemes, but utility is typically limited to a very small part of the development spectrum. Hence the concept of user-friendly, tailorable systems which can be adapted to specific project needs is a central theme in current requirements for development environments. Interface "friendliness" of the support environment arises from the use of good editors, command style, and informative diagnostics. Adaptability of the support environment is addressed by provision for common language processing which supports a variety of user oriented specification languages within a single architecturally consistent framework.

A major conclusion of our software engineering experience is that development environments such as an APSE must be usage and procedure oriented. This means that the techniques for using the development system, for manipulating its features, adding new capabilities and most importantly for representing its capabilities to the user, are more important than individual tools. While it

is true that the end utility programs accomplish the actual work of individual development aids, it is the framework of the environment that provides the control aspects. Hence file naming conventions, data base access techniques, command language or menu style, and system access structures which address multiple user classes determine the control procedures of environment operation. Consequently, the simple integration of plug-in tool components will not satisfy the spirit of support environment requirements.

Two critical aspects are missing in the plug-in concept: communication and extensibility. In systems as comprehensive as those to be dealt with in an APSE, multiple levels of users (managers, designers, programmers, administrators, etc.) demand a variety of communication levels to accommodate the multiple levels of expression represented in life cycle information constructs (requirements, structural design, behavioral design, logical design, program packages, data structures, etc.). A rich environment control structure which not only supports this concept but anticipates the need for future change is required. This leads to the second aspect, extensibility. The control process nature of the support environment will not be static because the dynamics of DoD applications and operations require a system which can respond to changes in organizational elements, and technology developments. The simple integration concept of "plug in" components cannot accommodate such needs because it is only extendable in the simplest of ways, namely the addition of individual tools.

In 1984 a Comprehensive Software Development Environment (CSDE) project was initiated at Martin Marietta. This project focuses previous research on methods, techniques, and tools for software system developments. The project will be developing a unified software, hardware, office facility, and data base management environment for use on Martin Marietta software development efforts. The CSDE concept was developed under the influence of the Ada STONEMAN and "METHODMAN" efforts.

The Martin Marietta view considers a proper Support Environment as a superstructure synergistically integrating present technological capabilities

while providing for change. Our approach with our own prototype environment concentrates on the environment control layer and its interfaces to the user community and host architecture. Individual tools are provided to process specific life-cycle phase data. These tools accommodate existing technologies particularly in the areas of high level specification languages, code transformation and maintenance. Moreover, emerging technology is also provided for, particularly with respect to Ada. Our concept recognizes the importance of Ada in the institutional processes relative to life-cycle economics. Hence the impact of Ada will be most apparent in the command language style of the Support Environment as well as the human interface aspect derived from the node model of the Common Ada program support environment Interface Set (CAIS).

Historical Perspective

In our software engineering research efforts at Martin Marietta we are currently building this prototype software development environment consisting of an integrated set of tools directed to supporting the full software life cycle from requirements to maintenance. Our approach to this activity has evolved after several years of research and development addressing software engineering problems, surveying and evaluating state-of-the-art efforts, and building experimental tools on an individual basis. A major effort to date was the establishment of the MEDSys concept. MEDSys stands for "Multilevel Expression Design System", and attempts to comply with a set of high level specification aid requirements that we defined.

- o Any successful support scheme will be an integrated set of (independent) tools which reflect the interactive nature of designing. Concepts phrased in relatively high levels of abstraction are incrementally refined into statements of increasing specificity.

- o Really successful tools must either be very simple to use and understand, or be firmly couched in an established methodology

apart from an automated representation.

- Computer aided tools are primarily directed to the process of software development, interim products being secondary and not a goal in and of themselves.

MEDSys was designed around three specification languages and their associated processors and analyzers, each of which can be used independently or in concert. The first two (requirements and structural design) were effectively accomplished by early 1979.

Each MEDSys component operates as an interactive system, driven by its own design data base. Once a design data base has been built, subsequent analysis processors can be called upon for evaluating the information accumulated - tantamount to automatically producing quality metrics. With multiple data bases, each representing a successive stage in the development life cycle, simple correlations on such quality metrics quickly evaluate their predictive accuracy.

In conjunction with this research and development effort, an auxiliary activity implemented a tool called the Automated Structured Analysis Processor (ASAP) which supports design in the Yourdan/DeMarco vein by formalizing the Data Flow Diagram and Process Description techniques with a specification language and corresponding data bases (data dictionary and "mini-spec"). Unlike MEDSys, ASAP is methodology specific.

We are now involved in the practical CSDE effort to further integrate these tools using the motivation of STONEMAN/AIE. In establishing this tool environment, we have investigated as many known similar efforts as possible, including:

- CSDP for RADC by CSC
- JSS for RADC/AFWAL by General Dynamics
- J73PSL for USAF/ASD by Softech

- Army ALS and Air Force APSE by Softech and Intermetrics
- ESPRESSO for NRC, Karlsruhe (Germany) by Ludewig & Eckert
- ISIS for NASA/LaRC by J. Berman (U of VA)

Additional influence on this effort stems from our ongoing Ada technology and training program coupled with our past involvement in the ALS and APSE contract evaluations and evaluation of the CAIS specification.

EVALUATING APSE EFFECTIVENESS IN DEVELOPING SOFTWARE

JAMES WINCHESTER
HUGHES AIRCRAFT COMPANY

APSE E&V WORKSHOP

EVALUATING APSE EFFECTIVENESS IN DEVELOPING SOFTWARE

by

Dr. James Winchester

Hughes Aircraft Company
P.O. Box 3310/618-M215
Fullerton, CA 92634

April 1984

APSE EVALUATION & VALIDATION (E & V) POSITION STATEMENT

Issue

The APSE E & V Task includes providing a means to evaluate the effectiveness of the User/APSE Interface.

An issue is whether this evaluation should include just application mechanics (e.g., ease of use, performance) or also evaluate the advantages of applying the APSE in terms of improvements in software productivity or quality. The E & V Task should provide a means to evaluate the overall effectiveness of applying an APSE.

A central focus of the Ada language and its associated support environment (APSE) is the perceived software development productivity and quality gains that can be realized in using such a system. An APSE can be considered as a collection of tools that support specification, design, and implementation techniques across the software development life cycle, from requirements definition through testing and maintenance.

The application of each tool to the software engineering job can provide some incremental improvement in productivity and quality.

The functional interaction between the APSE tools could create an environment that increases total productivity and quality improvements beyond a simple sum of incremental tool application improvements (the tools work together smoothly). Conversely, the functional interaction between the APSE tools could result in total productivity and quality improvements that are less than what would be expected from summing incremental tool application improvements.

Providing a means to evaluate the overall effectiveness of applying an APSE requires:

(1) a framework for evaluating life cycle phase unique transition techniques and the functional completeness of an APSE.

(2) a procedure and criteria for determining the advantages (e.g., productivity, higher reliability) of applying the APSE.

(3) a procedure and criteria for determing the cost of applying the APSE.

Transition Framework

A possible framework for evaluating the efficiency and effectiveness of transitioning between APSE tools is described in the following section.

As shown in Table K-5, a software development methodology consists of four related components: (1) a notation (graphic and textual) for representing the information required in the engineering activities during the phases of software development (requirements, design, etc.), (2) analysis techniques to determine the development information's level of compliance to certain desired criteria (e.g., completeness of requirements, consistency of design to requirements), (3) tools (particularly computer-aided) to support the manipulation of the notation and provide algorithmic checks for the analysis techniques, and (4) procedures to guide proper use of the notation, analysis techniques, and tools in moving from software requirements to implementation.

Most methodologies have concentrated on one phase of the software development life cycle, resulting in a set of unique notations, analysis techniques, tools and procedures. Complete software development life cycle methodologies can be created by integrating the phase unique methodologies via one or more components (see Table K-6).

The easiest type of integration is achieved through defining a comprehensive set of procedures that explains how to sequence a set of phase unique methodologies to achieve a complete methodology. With this type of integration, the engineer must understand many different notations and analysis techniques as well as use a noncohesive set of tools.

A further degree of integration is to link the computer-aided tools into a more cohesive set. This involves creating a common user interface and translation software to move information from one tool's database to another. Extended integration reduces the amount of time engineers need to complete their software development.

The most extensive type of methodology integration is achieved by developing notation and analysis techniques that can be applied across all software development phases. Given that such a canonical notation and analysis technique is defined, tools and procedures can be developed that are smoothly interfaced.

TABLE K-5. SOFTWARE METHODOLOGY COMPONENTS

Methodology Component	Example
• Notation	- Data Flow Diagram - Structure Chart
• Analysis Technique	- Data Flow Analysis - Structure Chart Analysis
• Automated Tools	- Automated Specification Analysis Tool (ASAT) - Automated Interactive Design Evaluation System (AIDES)
• Procedures	- Structured Analysis and Structured Design Procedures

TABLE K-6. POSSIBLE APPROACHES TO INTEGRATING METHODOLOGY COMPONENTS ACROSS LIFE CYCLE PHASES

Approach	Example	Advantage
• Notational/Analysis Techniques		
- *Comprehensive and cohesive* notation analysis techniques that can be used across all phases of the engineering life cycle	- *System* Architect Apprentice (SARA) - Integrated Definition (IDEF) methodology of the Integrated Computer Aided Manufacturing (ICAM) Project	- Easy to learn and apply - Most efficient type of complete methodology, but requires significant modification to existing engineering practice
• Automated Tools		
- Develop software to translate information from one tool's database to another	- Analysis and Design Interface Transforms (ADIT) which link a data flow tool (ASAT) to a structured design tool (AIDES)	- More efficient and effective than procedure only integration
• Procedural		
- Construct comprehensive procedures to link complete set of notation/techniques/tools	- Integrated Software Development Methodology (ISDM) Guidebook	- Relatively easy to develop

APSE Application Advantages Procedure and Criteria

The APSE application advantages procedure could include establishing standard software development "problems" to be specified, designed, implemented, tested etc., using the APSE under evaluation. If standard problems were defined to cover a wide variety of software applications (e.g., command and control, data base processing) then APSE evaluators could choose those most appropriate for their own domain.

Data could be collected during this APSE application, including such factors as number and type of errors surfaced, errors created during development phase transitions, and final software performance.

APSE Application Cost Procedure and Criteria

Procedures could be defined for collecting relevant cost data during APSE application to a standard evaluation problem. These data could include level of effort to transition information from the output of one phase unique tool to another, tool performance characteristics and hardware support requirements.

Summary

For the foreseeable future, there will be no consensus among users as to what unique set of tools will comprise an APSE. The E & V Task must address the issue of how to determine the overall effectiveness of an APSE in supporting the software development activities and not simply evaluating tool application mechanics.

U S GOVERNMENT PRINTING OFFICE 1986 - 559-007/10317

END

FILMED

6-85

DTIC